Functional Principal Components Analysis, Implementation and Applications

ABSCHLUSSARBEIT

zur Erlangung des akademischen Grades

Master of Science

(M.Sc.)

im Masterstudiengang Statistik

an der Wirtschaftwissenschaftlichen Fakultät

Humboldt-Universität zu Berlin

von

Mgr. Michal Benko B.Sc. Stat.

geboren am 31.07.1979 in Banská Bystrica

Gutachter:

Prof. Dr. Wolfgang Härdle
Prof. Dr. Bernd Rönz

eingereicht am December 20, 2004

Functional Principal Components Analysis, Implementation and Applications

A Master Thesis Presented

by

Michal Benko

(169567)

to

Prof. Dr. Wolfgang Härdle

CASE - Center of Applied Statistics and Economics

Humboldt University, Berlin

in partial fulfillment of the requirements

for the degree of

Master of Science

Berlin, December 20, 2004

Declaration of Authorship

I hereby confirm that I have authored this master thesis independently and without use of others than the indicated resources. All passages, which are literally or in general matter taken out of publications or other resources, are marked as such.

Michal Benko

Berlin, December 20, 2004

Acknowledgment

I would like to thank Professor Dr. Wolfgang Härdle and Professor Dr. Bernd Rönz for giving me the opportunity and motivation to write this thesis. I'm grateful to Jan Ulbricht and Szymon Borak for long discussions of the proper XploRe implementation. I'm also indebted to my family, without their support it would be impossible to finish this work.

Abstract

This master thesis discusses selected topics of Functional Data Analysis (FDA). FDA deals with the random variables (and process) with realizations in the (smooth) functional space. The first part of this thesis introduces the basic assumptions, notation and ideas of FDA, here we will mainly focus on the functional basis approach. The second chapter deals with the one of the most popular FDA technique – Functional Principal Components Analysis (FPCA).

FPCA is the functional analogue of the well known dimension reduction technique in the multivariate statistical analysis – search for the (pairwise orthogonal) linear transformations of the random vector with the maximal variance.

In the second part of this thesis we discuss the k-sample problem in the framework of the FPCA. As the starting point we use the Common Principal Components Modelling in the multivariate statistical analysis.

Apart from these theoretical considerations, the second main result of this thesis is the implementation of discussed FDA techniques in the statistical computing environment XploRe. Here we focus on the statistical macros (quantlets), graphical and plotting tools of functional data analysis.

Keywords: Functional Data Analysis, Regularization, Principal Components Analysis, Statistical Software

Contents

List of Figures

Notation

$\mathbb{R}$	set of real numbers
$\mathbb{R}^T$	T-dimensional real vector space
$\mathbb{N}$	set of natural numbers
$\mathcal{H}^*$	dual space of the abstract vector space $\mathcal{H}$
$\mathcal{D}\bullet$	differential operator (Newton-Leibnitz operator) on the corresponding functional space
$\mathcal{L}\bullet$	linear differential operator, linear combination of $\mathcal{D}^p, p = 1, \ldots, P$
$\mathrm{Ker}(\mathcal{L})$	kernel space of an operator $\mathcal{L}$
$\langle \bullet, \bullet \rangle$	scalar product defined on abstract (Hilbert) vector space
$\| \bullet \|^2$	norm defined on abstract (Banach) vector space
$[a, b)$	left closed, right opened interval a, b, similarly for closed, open intervals
$L^2(J)$	L^2 space on interval J - Lebesque square integrable functions on interval J, factorized with respect to the Lebesque measure.
$(\Omega, \mathcal{A}, \mathrm{P})$	probability space defined on the set Ω with the σ-algebra $\mathcal{A}$ and probability measure P
$\mathrm{span}(Z)$	span space, set of all possible linear combinations of the elements of set Z Z may by set of vectors or functions
$I(cond)$	Identificator function, if logical variable $cond = 1$ then $I(cond) = 1$ and 0 otherwise
$\mathrm{tr}(\mathbf{A})$	trace of an (square) matrix $\mathbf{A}$
$\det(\mathbf{A})$	determinant of an (square) matrix $\mathbf{A}$
$\mathrm{diag}(\mathbf{A})$	diagonal of an (square) matrix $\mathbf{A}$
$\mathbf{A}^{-1}$	inverse of an (nonsingular) matrix $\mathbf{A}$
exp	exponential function

1 Introduction

In the traditional multivariate framework the random objects are modelled through an T-dimensional random vector X. More formally an random vector is a measurable function $X : (\Omega, \mathcal{A}, P) \to (\mathbb{R}^T, \mathcal{B}^T)$, that maps an abstract probability space $(\Omega, \mathcal{A}, P)$ on the real measurable space $(\mathbb{R}^T, \mathcal{B}^T)$, where $\mathcal{B}^T$ are the Borel sets on $\mathbb{R}^T$. Observing N realizations of X we analyze the random vector X using the data set

$$\mathcal{X}_M \stackrel{\text{def}}{=} \{x_{i1}, x_{i2} \dots x_{iT}, i = 1, \dots, N\}.$$

In the functional data framework, objects are usually modelled as realizations of a stochastic process $X(t),\ t\ \in J$, where J is a bounded interval in $\mathbb{R}$. Thus, the set of functions

$$\mathcal{X}_f \stackrel{\text{def}}{=} \{x_i(t),\ i = 1, 2, \dots N,\ t \in J\}$$

represents the data set. More formally, the random function X in the functional data framework is modelled as a measurable function mapping $(\Omega, \mathcal{A}, \mathcal{P})$ into $(\mathcal{H}, \mathcal{B}_\mathcal{H})$, where $\mathcal{B}_\mathcal{H}$ is a Borel field on the functional Hilbert space $\mathcal{H}$.

We will mainly work with functional Hilbert space $L^2(J)$ with standard L^2 scalar product defined by

$$\langle f, g\rangle \stackrel{\text{def}}{=} \int_J f(t)g(t)dt,\ \text{for } \forall f, g \in L^2(J). \tag{1.1}$$

The corresponding $L^2(J)$ norm is determined by the scalar product (1.1) by $||f|| \stackrel{\text{def}}{=} \langle f, f\rangle^{1/2}$.

Moreover assume the existence of the mean, variance, covariance and functions of X, and denote these by $\mathrm{E}X(t)$, $\mathrm{Var}_X(t)$, $\mathrm{Cov}_X(s,t)$ and $\mathrm{Corr}_X(s,t)$ respectively:

$$\begin{aligned}
\mathrm{E}X(t) &\stackrel{\text{def}}{=} \mathrm{E}X(t),\ t \in J, \\
\mathrm{Var}_X(t) &\stackrel{\text{def}}{=} \mathrm{E}\{X(t) - \mathrm{E}X(t)\}^2,\ t \in J, \\
\mathrm{Cov}_X(s,t) &\stackrel{\text{def}}{=} \mathrm{E}\{X(s) - \mathrm{E}X(s)\}\{X(t) - \mathrm{E}X(t)\},\ s,t \in J, \\
\mathrm{Corr}_X(s,t) &\stackrel{\text{def}}{=} \frac{\mathrm{Cov}_X(s,t)}{\sqrt{\mathrm{Var}_X(s)\mathrm{Var}_X(t)}}.
\end{aligned}$$

Note that the $\mathrm{Corr}_X(s,t)$ is defined under the assumption $\mathrm{Var}_X(s), \mathrm{Var}_X(t) > 0$.

For the functional sample $x_i(t),\ i = 1, \ldots N$ we can define the estimators of the $\mathrm{E}X(t)$, $\mathrm{Var}_X(t)$, $\mathrm{Cov}_X(s,t)$ and $\mathrm{Corr}_X(s,t)$ in a straightforward way:

$$\begin{aligned}
\bar{x}(t) &= \tfrac{1}{N}\sum_{i=1}^{N} x_i(t), \\
\widehat{\mathrm{Var}}_X(t) &= \tfrac{1}{N-1}\sum_{i=1}^{N} \{x_i(t) - \bar{x}(t)\}^2, \\
\widehat{\mathrm{Cov}}_X(s,t) &= \tfrac{1}{N-1}\sum_{i=1}^{N} \{x_i(s) - \bar{x}(s)\}\{x_i(t) - \bar{x}(t)\}. \\
\widehat{\mathrm{Corr}}_X(s,t) &= \frac{\widehat{\mathrm{Cov}}_X(s,t)}{\sqrt{\widehat{\mathrm{Var}}_X(s)\widehat{\mathrm{Var}}_X(t)}}.
\end{aligned}$$

Dauxois, Pousse and Romain (1982) show that

$$||\mathrm{Cov}_X(s,t) - \widehat{\mathrm{Cov}}_X(s,t)|| \to 0, \text{ with probability one.}$$

This thesis is organized as follows: Chapter 2 introduces the most popular implementation of functional principal components – the functional basis expansion. The following Chapter 3 describes the software implementation of the statistical and graphical tools of FDA in the statistical software environment XploRe. In Chapter 4 we present the basic theory of the functional principal components, smoothed functional principal components, their implementation in XploRe and a practical application on the temperature data set. Chapter 5 discusses the common estimation of functional principal components. In the last chapter one can find the complete list of XploRe quantlets.

2 Functional Basis Expansion

In the previous chapter, we have presented the problem of statistical analysis of random functions. As we will see, from the theoretical point of view, the multivariate statistical concepts can be often introduced into the functional data analysis easily. However, in practice we are interested in the implementation of these techniques in fast computer algorithms, where a certain finite-dimensional representation of the analyzed functions is needed.

A popular way of FDA-implementation is to use a truncated functional basis expansion. More precisely, let us denote a functional basis on the interval J by $\{\theta_1, \theta_2, \dots, \}$ and assume that the functions x_i are approximated by the first L basis functions θ_l, $l = 1, 2, \dots L$

$$x_i(t) = \sum_{l=1}^{L} c_{il}\theta_l(t) = \mathbf{c}_i^\top \boldsymbol{\theta}(t), \tag{2.1}$$

where $\boldsymbol{\theta} = (\theta_1, \dots, \theta_L)^\top$ and $\mathbf{c}_i = (c_{i1}, \dots, c_{iL})^\top$. The first equal sign in (2.1) is not formally adequate – in practice we are just approximating the x_i. However, in order to keep the notation simple we will neglect this difference between the real $x_i(t)$ and its approximation.

In practice, the analysis of the functional objects will be implemented through the coefficient matrix

$$\mathbf{C} = \{c_{il},\ i = 1, \dots, N,\ l = 1, \dots, L\},$$

e.g. the mean, variance, covariance and correlation functions can be approximated by:

$$\begin{aligned}
\bar{x}(t) &= \bar{\mathbf{c}}^\top \boldsymbol{\theta}(t), \\
\widehat{\mathrm{Var}}_X(t) &= \boldsymbol{\theta}(t)^\top \mathrm{Cov}(\mathbf{C})\boldsymbol{\theta}(t), \\
\widehat{\mathrm{Cov}}_X(s,t) &= \boldsymbol{\theta}(s)^\top \mathrm{Cov}(\mathbf{C})\boldsymbol{\theta}(t), \\
\widehat{\mathrm{Corr}}_X(s,t) &= \frac{\widehat{\mathrm{Cov}}_X(s,t)}{\left\{\widehat{\mathrm{Var}}_X(t)\widehat{\mathrm{Var}}_X(s)\right\}^{1/2}}
\end{aligned}$$

where $\bar{\mathbf{c}}_l \stackrel{\text{def}}{=} \frac{1}{N}\sum_{i=1}^{N} c_{il},\ l = 1, \ldots, L$, $\mathrm{Cov}(\mathbf{C}) \stackrel{\text{def}}{=} \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{c}_i - \bar{\mathbf{c}})(\mathbf{c}_i - \bar{\mathbf{c}})^\top$.
The scalar product of two functions corresponds to:

$$\langle x_i, x_j \rangle \stackrel{\text{def}}{=} \int_J x_i(t)x_j(t)dt = \mathbf{c}_i^\top \mathbf{W}\mathbf{c}_j,$$

where

$$\mathbf{W} \stackrel{\text{def}}{=} \int_J \boldsymbol{\theta}(t)\boldsymbol{\theta}(t)^\top dt. \tag{2.2}$$

There are three prominent examples of functional bases: Fourier, Polynomial and B-Spline basis.

2.1 Fourier basis

A well known basis for periodic functions on the interval J is the Fourier basis, defined on J by

$$\theta_l(t) = \begin{cases} 1, & l = 0 \\ \sin(r\omega t), & l = 2r-1 \\ \cos(r\omega t), & l = 2r \end{cases}$$

where the frequency ω determines the period and the length of the interval $|J| = 2\pi/\omega$. The Fourier basis defined above can easily be transformed to an orthonormal basis, hence the scalar-product matrix in (2.2) is simply the identity matrix. The popularity of this basis is based partially on the possibility of fast coefficient calculation by the Fast Fourier Transformation (FFT) Algorithm. In XploRe environment one can use the quantlet `Fourierevalgd` for general case or the quantlet `fft` that performs the FFT Algorithm for the equidistant design.

2.2 Polynomial basis

The polynomial basis, appropriate for non-periodic functions is defined by

$$\theta_l(t) = (t-\omega)^k, k = 0, 1, \ldots, L-1$$

where ω is a shift parameter. The polynomial (or monomial) functions are easy to calculate for example by a simple recursion. However, the high order polynomials become too fluctuating especially in the boundaries of J. In XploRe one can use the quantlet `polyevalgd`.

2.3 B-Spline basis

A very popular functional basis for non-periodic data is the B-Spline basis. This basis is defined by the sequence of knots on the interval J and is roughly speaking a basis for piecewise polynomial functions of order K smoothly connected in the knots. More formally, the basis functions are

$$\theta_l(t) = B_{l,K}(t), l = 1, \ldots, m + k - 2 \tag{2.3}$$

where $B_{l,K}$ is l-th B-Spline of order K, for the non-decreasing sequence of knots $\{\tau_i\}_{i=1}^{m}$ defined by following recursion scheme:

$$B_{i,1}(t) = \begin{cases} 1, & \text{for } t \in [\tau_i, \tau_{i+1}] \\ 0, & \text{otherwise} \end{cases}$$

$$B_{i,k}(t) = \frac{t - \tau_i}{\tau_{i+k-1} - \tau_i} B_{i,k-1}(t) + \frac{\tau_{i+k} - t}{\tau_{i+k} - \tau_{i+1}} B_{i+1,k-1}(t)$$

for $i = 1, \ldots, m + k$, $k = 0, \ldots, K$. The number of the basis function will uniquely be defined by the B-spline order and the number of knots. The advantage of the B-spline basis is its flexibility, relatively easy evaluation of the basis functions and their derivatives. In XploRe one can use the quantlet `Bsplineevalgd`.

The detailed discussion of the implementation of the B-spline basis expansion in XploRe can be found in Ulbricht (2004).

2.4 Data set as basis

Let us briefly discuss an interesting special case – the use of the functions $x_i(t), i = 1, \ldots, N$ themselves as the basis. This directly implies that the coefficient matrix $\mathbf{C}$ is the identity matrix. Information about the data set $\mathcal{X}_f$ is "stored" in the matrix $\mathbf{W}$.

As we will show in next chapters this case has an direct application in practice, despite its pathological first-sight image.

2.5 Approximation and coefficient estimation

In practice, we observe the function values

$$\mathcal{X} \stackrel{\text{def}}{=} \{x_i(t_{i1}), x_i(t_{i2}), \ldots, x_i(t_{iT_i}),\ i = 1, \ldots, N\}$$

only on a discrete grid $\{t_{i1}, t_{i2}, \ldots, t_{iT_i}\} \in J$, where T_i are the numbers of design points for the i-th observation. In this case we may approximate the function $x_i(t)$ by functions $\theta_l(t), l = 1, \ldots L$ by minimizing some loss function, e.g. sum of squares. This approach is known as the least squares approximation. In case of using the data set as Basis we need to approximate integrals $\int x_l(t)x_k(t)dt$ by some numerical integration techniques.

A slightly different setup occurs if we assume that the data set is contaminated by some additive noise. A standard statistical approach is to assume that the data set consist of $\mathcal{X}_\varepsilon \stackrel{\text{def}}{=} \{Y_{ij}, j = 1, \ldots, T_i,\ i = 1, \ldots, N\}$, where

$$Y_{ij} = x_i(t_{ij}) + \varepsilon_{ij} \tag{2.4}$$

and ε_{ij} is the realization of a random variable with zero mean and variance function $\sigma_i^2(t)$, i.e. we are faced with the N regression problems. The estimated coefficient matrix $\mathbf{C}$ can be obtained by minimizing an appropriate loss function. The method of regularization by the roughness penalty can be applied. Defining the roughness penalty as the norm of an operator on the (Hilbert) space $\mathcal{H}$, $R \stackrel{\text{def}}{=} \| \mathcal{L} \|^2,\ \mathcal{L} \in \mathcal{H}^*$ we will minimize:

$$\sum_{j=1}^{T_i} \left(Y_{ij} - \mathbf{c_i}^\top \boldsymbol{\theta}(t_i j)\right)^2 + \alpha \| \mathcal{L}(\mathbf{c_i}^\top \boldsymbol{\theta}) \|^2 \tag{2.5}$$

where α is a parameter controling the degree of penalization. Clearly $\alpha = 0$ yields the least square regression. A popular example of the roughness penalty is $\mathcal{L} = \mathcal{D}^2$ where we penalize nonlinearity of the estimated function $\mathbf{c_i}^\top \boldsymbol{\theta}$. A more general appoach assumes $\mathcal{L}$ is a linear differential operator, i.e. $\mathcal{L} = a_1\mathcal{D}^1 + a_2\mathcal{D}^2 + \ldots + a_P\mathcal{D}^P$. The proper choice of the operator should have background in some additional information about the underlying function. Assume for example that $x_i \in \mathcal{V}$, $\mathcal{V} \subset \mathcal{H}$, than we should try to find an operator $\mathcal{L}$ so that $\mathrm{Ker}(\mathcal{L}) = \mathcal{V}$. Doing so we will penalize the coefficients that yield functions $\hat{x}_i = \mathbf{c_i}^\top \boldsymbol{\theta} \notin \mathcal{V}$.

Clearly, we can write $\mathcal{L}(\mathbf{c_i}^\top \boldsymbol{\theta}) = \mathbf{c_i}^\top \mathcal{L}(\boldsymbol{\theta})$, hence for implementation we need to be able just to calculate the function $\mathcal{L}(\theta_l)$.

Note that in the standard FDA setup, one assumes that the functions x_i are observed without additional error. This assumption is often violated in practice. Again in order to keep notation simple we will neglect the difference between the estimated and real coefficients. We can do so if we assume that the additional noise is of smaller order in compare to the variation of the functions x_i. However, from a statistical point of view we should keep this difference in mind.

One important question of practitioners is how many functions should be used in the basis expansion. Although, as stated by Ramsay and Silverman (1997), even a subjective selection of the smoothing parameter leads usually to the reasonable choice,

from a statistical point of view the automated (data driven selection) is needed. In the simplest case of e.g. Fourier basis without using additional regularization we need to set just the L. This can be done easily using Cross-Validation, Generalized Cross Validation or other similar criteria described in Härdle (1990) among others. Much more complicated is the case of B-splines – in practice we need to choose the knots sequence in addition to the number of functions. In some special applications the choice of knot points is naturally given by the underlying problem. One practical rule of thumb can be a good starting point: set at least 3 knots in the neighborhood of the "interesting" point of the function, e.g. around expected extreme-point or another change in the function.

An alternative approach may be applied in case we have additional information about the function of interest transformed into the roughness penalty $\| \mathcal{L} \|$.

The algorithm is as follows:

1. Use a "nested" model for the data set, i.e. use $L \approx$ number of observations. Using this basis directly would lead to a highly volatile estimator with a small (zero) bias.
2. Transform additional information about the function into the kernel of some appropriate linear differential operator.
3. Use the roughness penalty approach and estimate "smoothed" coefficients vector $\mathbf{c}_i$.

For the cubic B-splines basis, the first step corresponds to setting the knots into each design point. If we set $\mathcal{L} = \mathcal{D}^2$, we in fact penalize nonlinear functions, and obtain a special case of the very popular nonparametric technique – smoothing splines. In the third step of the algorithm we might easily set the smoothing parameter by Cross-Validation (CV) or Generalized Cross-Validation (GCV), for details see Hastie, et al (2002). This method is fully data driven for a given operator $\mathcal{L}$.

3 XploRe FDA Implementation

In this section we will deal with the XploRe implementation of FDA. The FDA-XploRe implementation is joint work with Jan Ulbricht (with clearly separated tasks). This fact is also indicated by the frequent reference to his thesis. The ideas already discussed by Ulbricht (2004) will be discussed very briefly, just in the extent that guarantees the understanding of further text. This is the case for the quantlets `createfdbasis`, `data2fd` and `evalfd` that have been programmed by Mr. Ulbricht. Our aim in this part was to implement basic statistical, descriptive and graphical tools for FDA and extend the existing system of quantlets in XploRe.

We implemented in XploRe general functional basis as a list object `basisfd` with following elements:

`fbname` - string, the name of the functional basis, supported are: `Fourier, Polynomial, Bspline`, that are mentioned above

`range` - vector of two elements, range of interval J

`param` - abstract set of elements, functional basis specific parameters that uniquely determine the functional basis `fbname`

`W` - optional parameter, the scalar product matrix $\mathbf{W}$

`penmat` - optional parameter, the penalty matrix

An functional object `fd` is similarly a list containing:

`basisfd` - object of type functional basis

`coef` - array of coefficients

These two objects can be created by following two quantlets.

`fdbasis = createfdbasis (fbname,range,param)`
creates the fdbasis object, on the interval `[range[1],range[2]] using parameters param`

`fd = data2fd (y, argvals, basisfd, Lfd, W, lambda)`
converts an array `y` of function values, or penalized regression with `Lfd` and `lambda` observed on an array `argvals` of argument values into a functional data object

In order to obtain an estimator of the mean function we can use:

`fdamean = fdamean(fdobject)`
creates a functional object with mean function from a functional object `fdobject`

The output of quantlet `fdamean` is a functional object. The functional object can be evaluated on a certain grid, we may create a XploRe graphical object or directly create a plot using:

`evalmat = evalfd (evalarg, fd, Lfd)`
evaluates a functional data object `fd`, operated by a linear differential operator `Lfd`, at argument values in an array `evalarg`

`gfd=grfd(fd,evalarg,Lfd,col,art,thick)`
creates a graphical object from the functional object `fd` operated by `Lfd`, using plotting grid `evalarg`, the line mask `col, art, thick` follow XploRe standards for `setmaskl`

`plotfd(fd,evalarg,Lfd,col,art,thick)`
plots the functional object `fd` operated by `Lfd`, using plotting grid `evalarg`, the line mask `col, art, thick` follow XploRe standards for `setmaskl`

For the variance, covariance and correlation functions we are faced with different situation because using functional basis expansion we need to evaluate a quadratic or bilinear form, which is not directly supported by the existing XploRe functional data objects. For this reason we designed quantlets for evaluating, creating of graphical objects and plotting of this functional data characteristics. The syntax of these quantlets is the following:

Variance function (one dimensional):

`fdvar=evalfdavar(evalarg,fdobject)`
evaluates the variance function of a functional data object `fd`, at argument values in an array `evalarg`

`gdfvar=grfdavar(fd,evalarg,col,art,thick)`
creates a graphical object from the variance function of a functional object `fd`, the line mask `col, art, thick` follow XploRe standards for `setmaskl`

`plotfdavar(fd,evalarg,col,art,thick)`
plots the variance function of a functional object `fd`, the line mask `col, art, thick` follow XploRe standards for `setmaskl`

Covariance and Correlation function (two dimensional functions) - surfaces:

`fdcov=evalfdacov(evalarg,fdobject)`
evaluates the cov function of a `fdobject` at the vector `evalarg` $\times$ `evalarg`

`fdcov=evalfdacorr(evalarg,fdobject)`
evaluates the corr function of a `fdobject` at the vector `evalarg` $\times$ `evalarg`

`gs=grfdacov(fdobject,evalarg,col)`
creates a graphical object with cov surface, using plotting grid `evalarg` × `evalarg` and color `col`

`gs=grfdacorr(fdobject,evalarg,col)`
creates a graphical object with corr surface, using plotting grid `evalarg` × `evalarg` and color `col`

`gs=plotfdacov(fdobject,evalarg,col,plot3Ds)`
plots the cov surface of `fdobject`, using plotting grid `evalarg` × `evalarg` and color `col`, if `plot3Ds`=1 the `plot3d` will be used

`gs=plotfdacorr(fdobject,evalarg,col,plot3Ds)`
plots the corr surface of `fdobject`, using plotting grid `evalarg` × `evalarg` and color `col`, if `plot3Ds`="3D" the `plot3d` will be used

The input parameters are:

`fd,fdobject`, lists, XploRe functional objects

`evalarg`, $m \times 1$ vector, grid for evaluation

`Lfd`, list, linear differential operator (LDO). This can be a scalar or a vector of integers which indicate the order of derivatives to obtain. In the case of an LDO with constant coefficients `Lfd` must be a (r × 2) matrix, where the first column contains the coefficients, the second the orders of derivatives. When to apply an LDO with variable coefficients `Lfd` must be an LDO object.

`col` vector of integers, color, parameter for `setmaskl`, default is 0

`art` vector of integers, art type, parameter for `setmaskl`, default is 1

`thick` vector of integers, thickness type, parameter for `setmaskl`, default is 2

`plot3Ds` string, if string is = "3D" plot3D quantlet will be used

The functions Cov, Corr are evaluated at the points $t_i, s_j \in$ evalarg. The usage of these quantlets will be illustrated by a simple example in the next section.

3.1 Temperature example

In this section we want to analyze a data set containing 35 weather stations in Canada, listed in the Table 3.1.

Arvida	Bagottvi	Calgary	Charlott	Churchil	Dawson
Edmonton	Frederic	Halifax	Inuvik	Iqaluit	Kamloops
London	Montreal	Ottawa	Princeal	Princege	Princeru
Quebec	Regina	Resolute	Scheffer	Sherbroo	Stjohns
Sydney	Thepas	Thunderb	Toronto	Uraniumc	Vancouvr
Victoria	Whitehor	Winnipeg	Yarmouth	Yellowkn	

Table 3.1: Names of the Canadian weather stations.

This data set is taken from Ramsay and Silverman (1997), the data with the description can be found in the online database MD*Base. We choose this example as a "guinea pig" data set that illustrates the defined concepts and the use of the implemented quantlets and gives the possibilities of comparison.

Due to the cyclical behavior of the temperature during years it is usual in comparable studies to assume the temperature functions to be periodic. Thus we may employ the Fourier basis functions. Figure 3.1 shows estimated temperature functions using 31 Fourier functions.

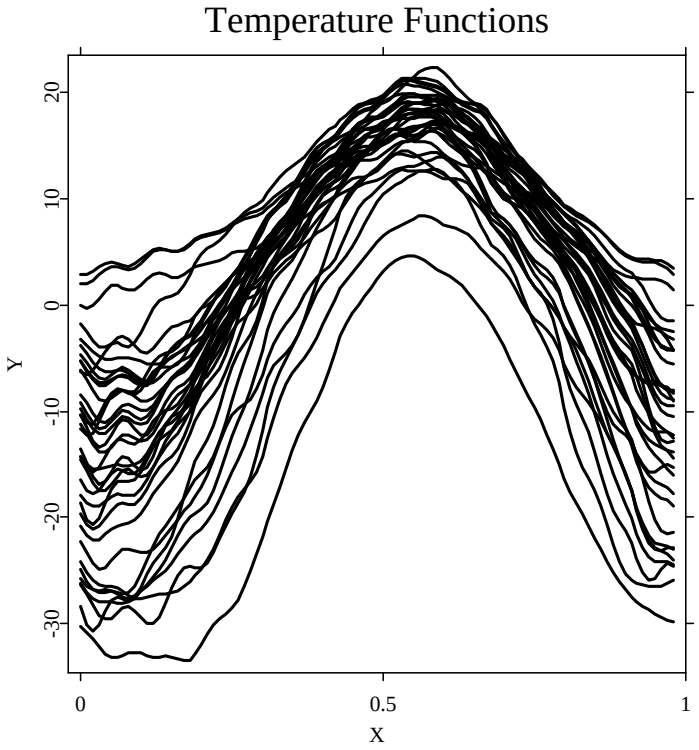

Figure 3.1: Example of functional data, temperatures measured by Canadian weather stations listed in the Table 3.1.

FDAmscTempf.xpl

For our Temperature data set we obtained the following functions that correspond to the basic statics in multivariate framework:

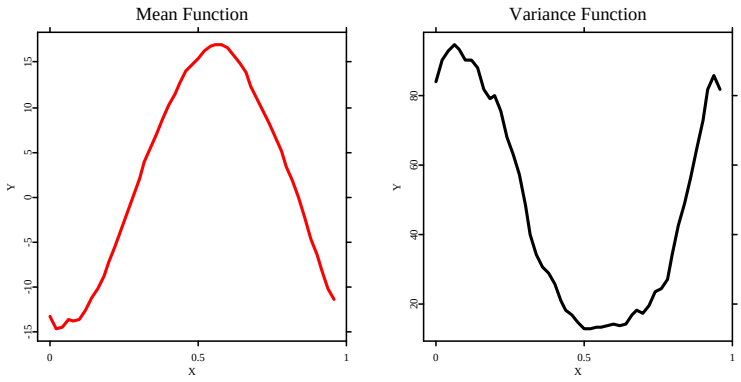

Figure 3.2: Mean and variance function (temperature).

FDAmscMeanVar.xpl

Looking at the functions in the Figures 3.2 and 3.4 we observe the mean function with a bump in the summer, which is not surprising – temperatures are higher in summer than in winter. Another fact may be, however, not so expected: in the winter we can observe higher variance than in the summer. This fact can possibly be explained by the higher impact of the geographical differences on the winter temperature than on the summer temperature. Looking at the correlation function, plotted in the Figure 3.4 we can see that the autumn temperatures seems to be higher correlated with the spring temperatures than winter and summer temperatures.

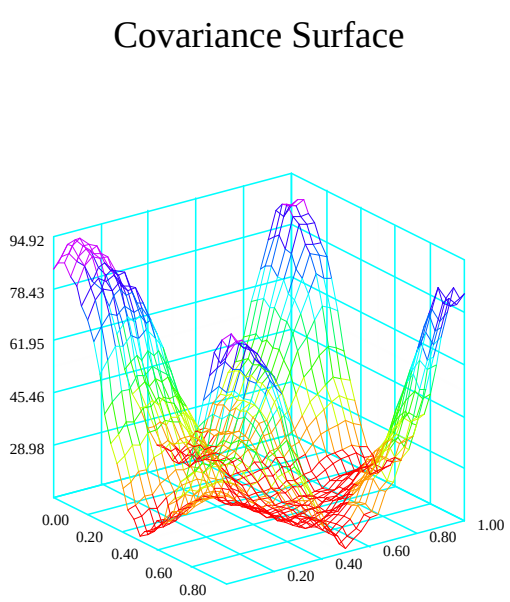

Figure 3.3: Covariance function (temperature). The plotted surface is the linear interpolation of grid on $[0, 1]$ and step $1/30$.

FDAmscCov.xpl

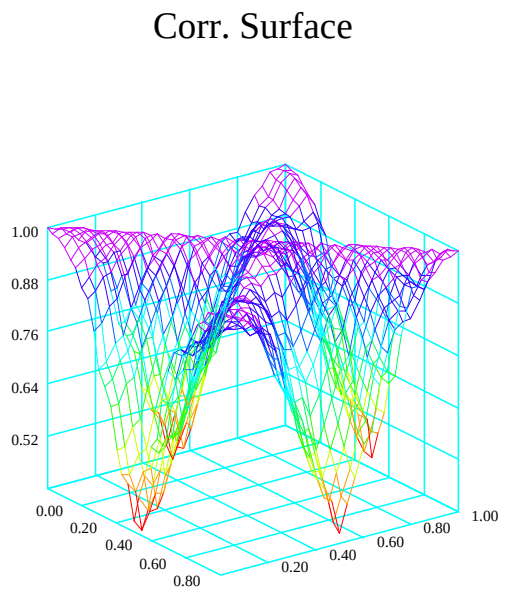

Figure 3.4: Correlation function (temperature). The plotted surface is the linear interpolation of grid on $[0, 1]$ and step $1/30$.

`FDAmscCorr.xpl`

4 Functional Principal Components

Principal Components Analysis (PCA) yields dimension reduction in the multivariate framework. The idea is to find the normalized weight vectors $\gamma_m \in \mathbb{R}^T$ for which the linear transformations of a T-dimensional random vector $\mathbf{x}$:

$$\beta_m = \gamma_m^\top(\mathbf{x} - \mathrm{E}\mathbf{x}) = \ \langle \gamma_m, \mathbf{x} - \mathrm{E}\mathbf{x} \rangle, \ m = 1, \ldots, T, \tag{4.1}$$

have maximal variance subject to:

$$\gamma_l^\top \gamma_m = \ \langle \gamma_l, \gamma_m \rangle \ = I(l = m) \text{ for } l \leq m.$$

The problem is solved by the means of the Jordan spectral decomposition of the covariance matrix, Härdle and Simar (2003), p. 63.

In Functional Principal Components Analysis (FPCA) the dimension reduction can be achieved via the same route: Find orthonormal weight functions $\gamma_1, \gamma_2, \ldots$, such that the variance of the linear transformation is maximal.

The weight functions satisfy:

$$\begin{aligned} ||\gamma_m||^2 &= \int \gamma_m(t)^2 dt = 1, \\ \langle \gamma_l, \gamma_m \rangle &= \int \gamma_l(t)\gamma_m(t)dt = 0, \ l \neq m. \end{aligned}$$

The linear combination is:

$$\beta_m = \ \langle \gamma_m, X - \mathrm{E}X \rangle \ = \int \gamma_m(t)\{X(t) - \mathrm{E}X(t)\}dt, \tag{4.2}$$

and the desired weight functions solve:

$$\underset{\langle \gamma_l, \gamma_m \rangle = I(l=m), l \leq m}{\arg\max} \mathrm{Var}\langle \gamma_m, X \rangle, \tag{4.3}$$

or equivalently:

$$\underset{\langle \gamma_l, \gamma_m \rangle = I(l=m), l \leq m}{\arg\max} \int \int \gamma_m(s)\mathrm{Cov}(s,t)\gamma_m(t)dsdt.$$

The solution is obtained by solving the Fredholm functional eigenequation

$$\int \mathrm{Cov}(s,t)\gamma(t)dt = \lambda\gamma(s). \tag{4.4}$$

The eigenfunctions $\gamma_1, \gamma_2, \ldots$, sorted with respect to the corresponding eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots$ solve the FPCA problem (4.3). The following link between eigenvalues and eigenfunctions holds:

$$\lambda_m = \mathrm{Var}(\beta_m) = \mathrm{Var}\left[\int \gamma_m(t)\{X(t) - \mathrm{E}X(t)\}dt\right] = \int\int \gamma_m(s)\mathrm{Cov}(s,t)\gamma_m(t)dsdt.$$

In the sampling problem, the unknown covariance function $\mathrm{Cov}(s,t)$ needs to be replaced by the sample covariance function $\widehat{\mathrm{Cov}}(s,t)$. Dauxois, Pousse and Romain (1982) show that the eigenfunctions and eigenvalues are consistent estimators for λ_m and γ_m and derive some asymptotic results for these estimators.

4.1 Implementation

In this section we will present three possibilities of implementation of the functional PCA. We will start with a simple discretization technique. In the second step we will focus more on the implementation by the basis expansion, including the application on the temperature data set.

4.1.1 Discretization

One possibility of calculating the functional PCA is simply to perform the multivariate PCA on a dense grid $\{t_1, \ldots, t_T\}$, obtain the eigenvectors $(\widehat{\gamma}_j(t_i),\ i = 1, \ldots, T)^\top$ for $j = 1, \ldots, r$ and estimate the coefficients of eigenvectors $b_1, \ldots, b_r$. The implementation is very simple since the routines of the multivariate PCA or just matrix spectral analysis are needed. However, in the next section we will present a implementation method that corresponds more to the functional nature of the underlying problem. Secondly, the estimated eigenfunctions are not necessarily orthonormal in the functional sense, due to the dependence of the integrals on the length of the interval and of the vector scalar product of the discretized functions on the number of discretization points T . Thus an additional correction is needed. A simple way is to orthonormalize the coefficient with respect to the matrix $\mathbf{W}$ using Gramm-Schmidt procedure.

4.1.2 Basis expansion

Suppose that the weight function γ has the expansion

$$\gamma = \sum_{l=1}^{L} \mathbf{b}_l \theta_l = \boldsymbol{\theta}^\top \mathbf{b}.$$

Using this notation we can rewrite the left hand side of eigenequation (4.4):

$$\begin{aligned} \int \mathrm{Cov}(s,t)\gamma(t)dt &= \int \boldsymbol{\theta}(s)^\top \mathrm{Cov}(\mathbf{C})\boldsymbol{\theta}(t)\boldsymbol{\theta}(t)^\top \mathbf{b} dt \\ &= \boldsymbol{\theta}^\top \mathrm{Cov}(\mathbf{C})\mathbf{W}\mathbf{b}, \end{aligned}$$

so that:

$$\mathrm{Cov}(\mathbf{C})\mathbf{W}\mathbf{b} = \lambda \mathbf{b}.$$

The functional scalar product $\langle \gamma_l, \gamma_k \rangle$ corresponds to $\mathbf{b}_l^\top \mathbf{W} \mathbf{b}_k$ in the truncated basis framework, in the sense that if two functions γ_l and γ_k are orthogonal, the corresponding coefficient vectors $\mathbf{b}_l, \mathbf{b}_k$ satisfy $\mathbf{b}_l^\top \mathbf{W} \mathbf{b}_k = 0$. Matrix $\mathbf{W}$ is symmetric by definition, thus, defining $\mathbf{u} = \mathbf{W}^{1/2}\mathbf{b}$, one needs to solve finally a symmetric eigenvalue problem:

$$\mathbf{W}^{1/2}\mathrm{Cov}(\mathbf{C})\mathbf{W}^{1/2}\mathbf{u} = \lambda \mathbf{u},$$

and to compute the inverse transformation $\mathbf{b} = \mathbf{W}^{-1/2}\mathbf{u}$. For the orthonormal functional basis (i.e. also for Fourier basis) $\mathbf{W} = \mathbf{I}$, i.e. the problem of FPCA is reduced to the multivariate PCA performed on the matrix $\mathbf{C}$.

Algorithm

1. calculate $\mathbf{C}$ and $\mathbf{W}$
2. using Cholesky decomposition calculate $\mathbf{W}^{1/2}$
3. use symmetric matrix eigenvalue routine and obtain eigenvalues and eigenvectors ($\mathbf{u}$) of $\mathbf{W}^{1/2}\mathrm{Cov}(\mathbf{C})\mathbf{W}^{1/2}$
4. calculate $\mathbf{b} = \mathbf{W}^{-1/2}\mathbf{u}$

Notice: The estimated coefficient of eigenfunctions are orthonormal with respect to the scalar product $\mathbf{b}_i^\top \mathbf{W} \mathbf{b}_j$, however, due to some numerical errors there can by small deviances.

Temperature example

For our temperature example we obtained the weight functions (eigenfunctions) displayed in the Figure 4.1, using the same setup as in the previous section (31 Fourier functions), with the following variance proportions (eigenvalues):

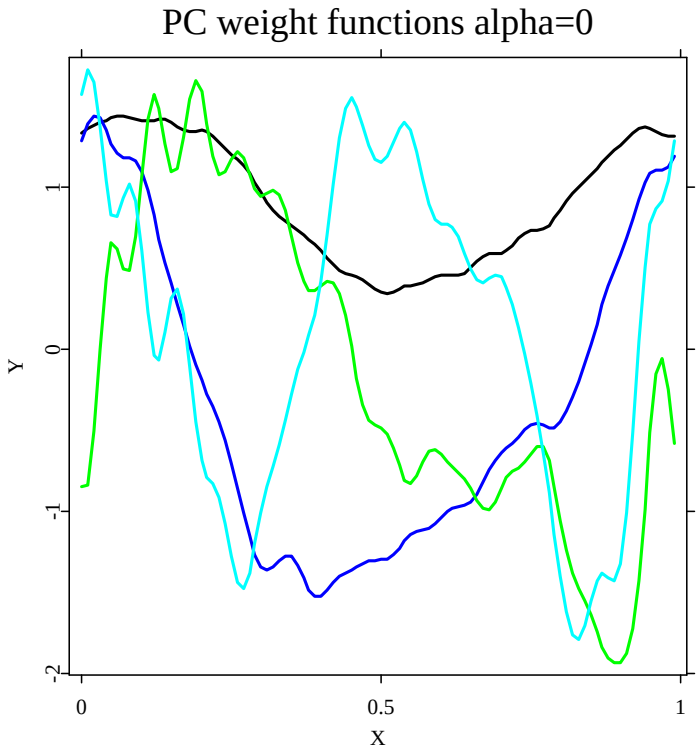

Figure 4.1: Weight functions for temperature data set.

FDAmscTempPCA.xpl

```
[1,]  0.88671
[2,]  0.084837
[3,]  0.019451
[4,]  0.0052221
```

The first eigenfunction (black curve) can be explained as a weighted level of the temperature over the year with a high weighting of the winter temperatures, this eigenfunction explains 89% of variation. The second eigenfunction (blue curve) has different signs for winter and summer and explains 8% of variation. The third eigenfunction (green curve) changes the sign in a similar way for autumn and spring and explains 2% of variation and the fourth function (cyan curve) could be explained as changes of seasons and explains 0.5% of variation.

However, the eigenfunctions are rough (non-smooth). This roughness is caused by sampling variance or by the observation noise and flexibility of used functional basis. In the Section 4.2 we will discuss the method of smoothing the eigenfunctions in order to get more stable and better interpretable results.

Another possibility of interpreting the result is the plot of the estimated principal scores, sample analogue of β_m:

$$\widehat{\beta}_{im} \stackrel{\text{def}}{=} \langle \widehat{\gamma}_m, x_i - \bar{x} \rangle = \mathbf{b}_m^\top \mathbf{W}(\mathbf{c}_i - \bar{\mathbf{c}})$$

The estimated principal scores are plotted in the Figure 4.2, the abbreviations of the names, listed in Table 4.1.2 have been used (compare with Table 3.1).

arv	bag	cal	cha	chu	daw
edm	fre	hal	inu	iqa	kam
lon	mon	ott	pri	prig	pru
que	reg	res	sch	she	stj
syd	the	thu	tor	ura	van
vict	whit	win	yar	yel	

Table 4.1: Abbreviations of the names of the Canadian weather stations.

A simple observation is that `res` is a kind of outlier, with high loading of second component. The `res` stands for Resolute, using similar arguments as in Ramsay and Silverman (1997). Resolute is known for having a very cold winter relative (PC1) to the other weather stations but only a small difference between summer and winter temperatures (PC 2). This corresponds to our interpretation of the first two eigenfunctions.

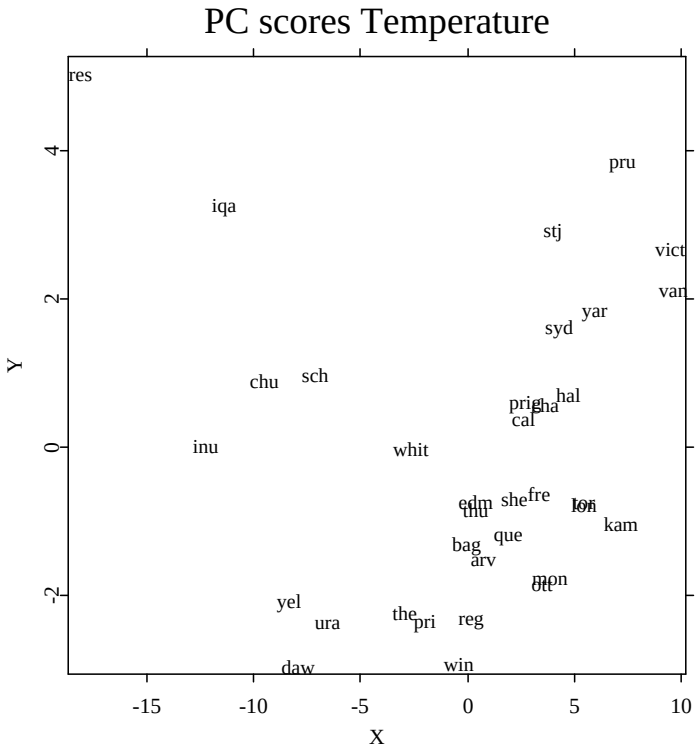

Figure 4.2: Principal scores for temperature data set

FDAmscTempPCA.xpl

4.1.3 Data set as basis

As announced in the previous section, if we use the data set $\mathcal{X}_f$ as the basis we need to estimate the matrix $\mathbf{W}$ rather than the coefficient matrix $\mathbf{C}$. We will show how to use this concept in the functional principal components analysis.

Estimation procedure is based on the Karhunen-Loèvy decomposition – we use eigen-

functions as factor functions in the model of following type:

$$x_i = \bar{x} + \sum_{j=1}^{K} \beta_{ij}\gamma_j, \tag{4.5}$$

recall, $\bar{x}$ is the sample mean, $\bar{x} = 1/N \sum_{i=1}^{N} x_i$. The K is the number of nonzero eigenvalues of the (empirical) covariance operator.

$$\mathcal{C}_N(\xi) = \frac{1}{N} \sum_{i=1}^{N} \langle x_i - \bar{x}, \xi \rangle (x_i - \bar{x}) \tag{4.6}$$

This is the sample version of the covariance operator used in (4.15). Lets us denote the eigenvalues, eigenvectors of $\mathcal{C}_N$ by $\lambda_1^{\mathcal{C}_N}, \lambda_2^{\mathcal{C}_N} \ldots, \gamma_1^{\mathcal{C}_N}, \gamma_2^{\mathcal{C}_N} \ldots$, and the principal scores $\beta_{ir}^{\mathcal{C}_N} \stackrel{\text{def}}{=} \langle \gamma_r^{\mathcal{C}_N}, x_i - \bar{x} \rangle$.

Using this notation model (4.5) is based on the known fact, that the first L eigenfunctions of the empirical covariance operator, ordered by the corresponding eigenvalues, construct the "best empirical basis" in the integrated square error sense, i.e. the residual integrated square error:

$$\rho(m_1, \ldots, m_L) = \sum_{i=1}^{N} \parallel x_i - \bar{x} - \sum_{j=1}^{L} \beta_{ij} m_j(t) \parallel^2 \tag{4.7}$$

is minimized with respect to all L orthogonal functions $m_j \in L^2$, $j = 1, \ldots, L$ by setting $m_j(t) \equiv \gamma_j^{\mathcal{C}_N}(t), t \in J$. In the estimation procedure, we follow the idea of Kneip and Utikal (2001), introduced for the case where $x_i(t)$ is a density function. Instead of estimating the eigenfunction $\gamma_j^{\mathcal{C}_N}$ by discretization or functional basis expansion of x_i we focus on the spectral analysis of matrix

$$M_{lk} = \langle x_l - \bar{x}, x_k - \bar{x} \rangle. \tag{4.8}$$

This procedure is motivated by the following two facts: firstly all nonzero eigenvalues of the empirical covariance operator $\mathcal{C}_N$ and the eigenvalues of the matrix M denoted by $\lambda_1^M, \lambda_2^M, \ldots$ are related as follows: $\lambda_r^M = N.\lambda_r^{\mathcal{C}_N}$. Secondly, for the eigenvectors of M denoted by $p_1, p_2, \ldots$ and principal scores $\beta_{jr}^{\mathcal{C}_N}$ holds:

$$\beta_{jr}^{\mathcal{C}_N} = \sqrt{\lambda_r^M} p_{jr} \tag{4.9}$$

and

$$\gamma_r^{\mathcal{C}_N} = \left(\sqrt{\lambda_r^M}\right)^{-1} \sum_{i=1}^{N} p_{ir} \left(x_i - \bar{x}\right) = \left(\sqrt{\lambda_r^M}\right)^{-1} \sum_{i=1}^{N} p_{ir} x_i = \frac{\sum_{i=1}^{N} x_i \beta_{ir}^{\mathcal{C}_N}}{\sum_{i=1}^{N} (\beta_{ir}^{\mathcal{C}_N})^2}. \tag{4.10}$$

The estimation procedure follows now in two steps. First we estimate the M by an appropriate estimator $\widehat{M}$ and in the second step we use (4.10) to obtain the estimators $\hat{\beta}_{ir}, \hat{\gamma}_r$ of principal scores and eigenfunctions $\beta_{ir}^{\mathcal{C}_N}, \gamma_r^{\mathcal{C}_N}$.

There are several aspects of this technique that need to be mentioned. First we need to mention that we virtually separate the spectral analysis and move it into the (feature) space of scalar products. Using appropriate robust estimators in (4.10) we may achieve robust estimators of eigenfunctions. In another hand in the general case of unequal designs the estimation of the scalar products can be complicated. The deeper discussion, for the special case of the density functions can be found in above mentioned Kneip and Utikal (2001), discussion of using this idea in the regression case can be found in Benko and Kneip (2005).

4.2 Smoothed principal components analysis

As we see in the Figure 4.1, the resulting eigenfunctions are often very rough. Smoothing them could result in more stable and better interpretable results. Here we apply a popular approach known as the roughness penalty. The downside of this technique is that we loose orthogonality in the L^2 sense.

Assume that the underlying eigenfunctions have a continuous and square-integrable second derivative. Recall that $\mathcal{D}\gamma = \gamma'(t)$ is the differential operator and define the roughness penalty by $\Psi(\gamma) = ||\mathcal{D}^2\gamma||^2$. Moreover, suppose that γ_m has square-integrable derivatives up to degree four and that the second and the third derivative satisfy one of the following conditions:

1. $\mathcal{D}^2\gamma$, $\mathcal{D}^3\gamma$ are zero at the ends of the interval J
2. the periodicity boundary conditions of γ,$\mathcal{D}\gamma$, $\mathcal{D}^2\gamma$ and $\mathcal{D}^3\gamma$ on J.

Then we can rewrite the roughness penalty in the following way:

$$\begin{aligned} ||\mathcal{D}^2\gamma||^2 &= \int \mathcal{D}^2\gamma(s)\mathcal{D}^2\gamma(s)ds \\ &= \mathcal{D}\gamma(u)\mathcal{D}^2\gamma(u) - \mathcal{D}\gamma(d)\mathcal{D}^2\gamma(d) - \int \mathcal{D}\gamma(s)\mathcal{D}^3\gamma(s)ds \qquad (4.11) \\ &= \gamma(u)\mathcal{D}^3\gamma(u) - \gamma(d)\mathcal{D}^3\gamma(d) - \int \gamma(s)\mathcal{D}^4\gamma(s)ds \qquad (4.12) \\ &= \langle\gamma, \mathcal{D}^4\gamma\rangle, \qquad (4.13) \end{aligned}$$

where d and u are the boundaries of the interval J and the first two elements in (4.11) and (4.12) are both zero under both conditions mentioned above.

Given a principal component function γ, with norm $||\gamma||^2 = 1$, we can penalize the sample variance of the principal component by dividing it by $1 + \alpha\langle\gamma, \mathcal{D}^4\gamma\rangle$:

$$PCAPV = \frac{\int\int \gamma(s)\widehat{\mathrm{Cov}}(s,t)\gamma(t)dsdt}{\int \gamma(t)(\mathcal{I} + \alpha\mathcal{D}^4)\gamma(t)dt}, \tag{4.14}$$

where $\mathcal{I}$ denotes the identity operator. The maximum of the penalized sample variance of the principal component (PCAPV) is an eigenfunction γ corresponding to the largest eigenvalue of the generalized eigenequation:

$$\int \widehat{\mathrm{Cov}}(s,t)\gamma(t)dt = \lambda(\mathcal{I} + \alpha\mathcal{D}^4)\gamma(s). \tag{4.15}$$

As already mentioned above, the resulting weight functions are no longer orthonormal in the L^2 sense. Since the weight functions are used as smoothed estimators of principal components functions, we need to rescale them to satisfy $||\gamma_l||^2 = 1$. The weight functions γ_l can be interpreted as orthogonal in modified scalar product of the Sobolev type

$$(f, g) \stackrel{\text{def}}{=} \langle f, g\rangle + \alpha\langle\mathcal{D}^2 f, \mathcal{D}^2 g\rangle.$$

A more extended theoretical discussion can be found in Silverman (1991).

4.2.1 Implementation using basis expansion

Define $\mathbf{K}$ to be a matrix whose elements are $\langle D^2\theta_j, D^2\theta_k\rangle$. Then the generalized eigenequation (4.15) can be transformed to:

$$\mathbf{W}\,\mathrm{Cov}(\mathbf{C})\mathbf{W}\mathbf{u} = \lambda(\mathbf{W} + \alpha\mathbf{K})\mathbf{u}. \tag{4.16}$$

Finding matrix L for that holds: $\mathbf{L}\mathbf{L}^\top = \mathbf{W} + \alpha\mathbf{K}$ and defining $\mathbf{S} = \mathbf{L}^{-1}$ we can rewrite (4.16) into:

$$\{\mathbf{S}\mathbf{W}\,\mathrm{Cov}(\mathbf{C})\mathbf{W}\mathbf{S}^\top\}(\mathbf{L}^\top\mathbf{u}) = \lambda\mathbf{L}^\top\mathbf{u}.$$

Algorithm

1. calculate $\mathbf{C}$ and $\mathbf{W}$
2. using Cholesky decomposition calculate $\mathbf{L}$ and their inverse $\mathbf{L}^{-1}$
3. use symmetrical matrix eigenvalue-eigenvector routine and obtain eigenvalues and eigenvectors ($\mathbf{u}$) of $\mathbf{S}\mathbf{W}\,\mathrm{Cov}(\mathbf{C})\mathbf{W}\mathbf{S}^\top$

4. calculate $\mathbf{b} = \mathbf{L}^{-1}\mathbf{u}$
5. renormalize $\mathbf{b}$ with respect to matrix $\mathbf{W}$, so that $\mathbf{b}^\top \mathbf{W}\mathbf{b} = 1$

If we are looking at the first K eigenfunctions as the best empirical basis for the functional observations $\mathcal{X}_f$, we may also re-orthonormalize coefficients $\mathbf{b}_j$ with respect to matrix $\mathbf{W}$, using Gramm-Schmidt procedure.

In this chapter we have presented the case with roughness penalty $\| \mathcal{D}^2\gamma \|$ similarly we could consider a more general case with rougness penalty $\| \mathcal{L}\gamma \|$.

4.2.2 XploRe implementation

The smoothed functional PCA is implemented in the XploRe via the quantlet `fdaspca`, with the following syntax:

```
{fpcaresult,values,varprop,scores} =fdaspca(fdobject{,lambda,
                                            lfd,npc,norm})
      performs the smoothed functional PCA
```

The input parameters:

`fdobject` list, functional object with N replications

`lambda` scalar, the smoothing parameter, default is 0

`npc` scalar, the number of (generalized) eigenfunctions, default is 4

`norm` string, normalization type, if norm="orthonorm" fpcaresult.coef are orthonormalized (with respect to the basis penalty matrix), if norm="norm" the coefficients are renormalized to norm=1, default is "no" – no normalization

The outputvariable

`fpcaresult` list, functional object

`values` `npc` $\times$ 1 vector, the (generalized) eigenvalues

`varprop` `npc` $\times$ 1 vector, the proportion of variance explained by the eigenfunctions

`scores` `N` $\times$ `npc` matrix, the principal components scores

The plot of the estimate principal scores, Figure 4.2, can be obtained easily from the output of quantlet `fdaspca`, using the command `setmask`:

```
library("plot")
library("fda")
y = read ("dailtemp.dat")
tvec=#(1:365)/365
fdb = createfdbasis ("fourier", #(0,1),31,1)
fdtempf31=data2fd(y,tvec,fdb)
fdapc=fdaspca(fdtempf31,0.00000,2,4)
TempdiSC=createdisplay(1,1)
grtempf=grfd(fdapc.fpcaresult,grid(0,1/100,100),0,#(0,1,2,3))
sc=fdapc.scores[,1:2]
labels=#("arv", "bag", "cal", "cha", "chu", "daw","edm", "fre",
"hal", "inu", "iqa", "kam","lon", "mon", "ott", "pri", "prig",
"pru","que", "reg", "res", "sch", "she", "stj","syd", "the",
"thu", "tor", "ura", "van","vict", "whit", "win", "yar", "yel")
setmaskt(sc,labels)
setmaskp(sc,0,0,0)
show(TempdiSC,1,1,sc)
setgopt(TempdiSC,1,1,"title","PC scores Temperature","border",0)
```

4.2.3 Temperature example

Performing the idea of Smoothed Functional PCA (SPCA) on the temperature data set, we obtained with the same setup as in previous sections (31 Fourier functions and $\alpha = 10^{-6}$) the weight functions plotted in Figure 4.3. We can observe that the high frequency variance is smoothed out and the interpretation of the eigenfunctions is more obvious.

We will illustrate one practical problem, the complementarity of the L and α, on following example: We obtain smoother eigenfunctions by decreasing the number of functions L or increasing the smoothing parameter α. In our temperature example we can obtain the degree of smoothness similar to using 31 Fourier functions and $\alpha = 10^{-6}$ with 11 Fourier series and $\alpha = 0$, as plotted in the Figure 4.4.

Clearly the two sets of eigenfunctions are not same. In order to avoid this difficulty we propose a similar procedure as in the Section 2.5:

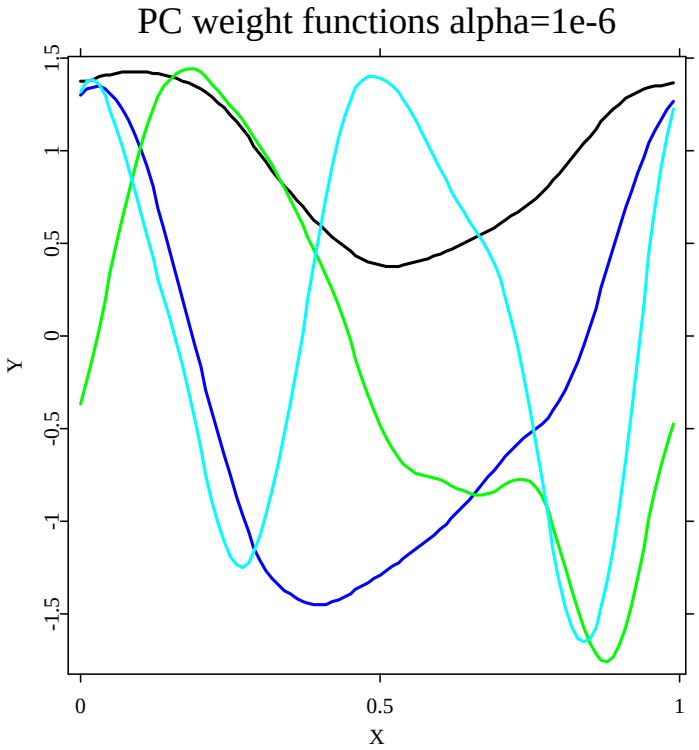

Figure 4.3: Weight functions for temperature data set, using $\alpha = 10^{-6}$ and L=31.
FDAmsdTempSPCA.xpl

1. Use set $L \approx N$, with LS regression (without penalization), i.e. undersmooth or even just interpolate the observed data set $\mathcal{X}$
2. Set appropriate α to smooth directly eigenfunctions.

For the choice of appropriate α one can use the technique described in Section 4.2.4. This procedure is statistically "clean" but computationally intensive.

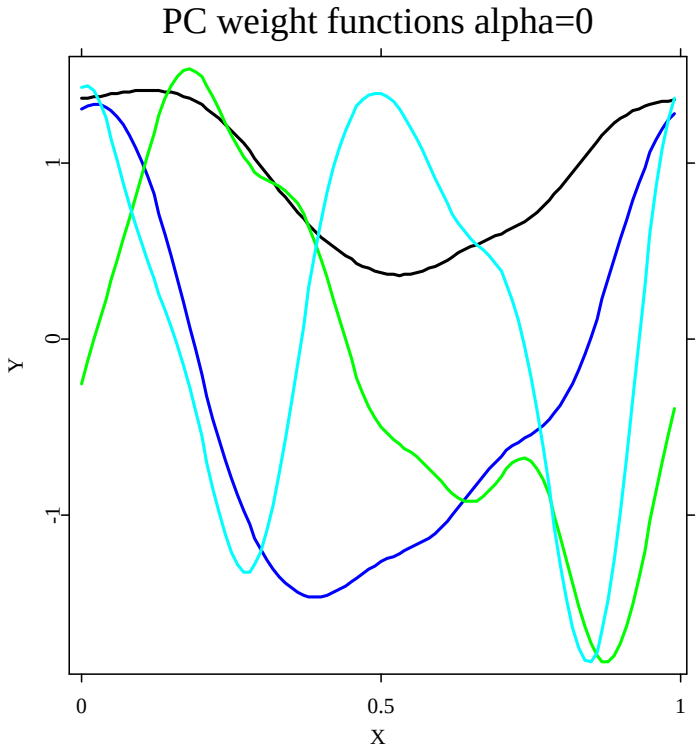

Figure 4.4: Weight functions for temperature data set, using $\alpha = 0$ and L=11.
FDAmscTempPCA11.xpl

4.2.4 Choice of the smoothing parameter

The natural question that appears in SPCA is the selection of the smoothing parameter α. A popular and intuitive solution to this problem is Cross-Validation. This approach is based on the fact that the eigenfunctions $\gamma_1, \gamma_2, \ldots, \gamma_M$ can be seen as an optimal empirical basis as discussed in the Section 4.1.3. In order to keep notation simple, assume that the sample mean $1/N \sum_i^N x_i = 0$, this can be done simply by subtracting the sample mean from functions x_i. (Using functional basis we subtract the sample mean of the coefficients $\bar{\mathbf{c}}$ from the coefficients $\mathbf{c}_i$.) Let $\mathbf{G}$ be the scalar product matrix

of the first M eigenfunctions, i.e. $\mathbf{G}_{ij} = \langle \gamma_i, \gamma_j \rangle$. For a function x, define the part of x orthogonal to the subspace spanned by the basis $\gamma_1, \gamma_2, \ldots, \gamma_M$ as

$$\zeta_M = x - \sum_{i=1}^{M} \sum_{j=1}^{M} (\mathbf{G}^{-1})_{ij} \langle \gamma_i, x \rangle \gamma_j.$$

Recall that for the SPCA the eigenfunctions are no longer orthonormal, thus $\mathbf{G} \neq \mathbf{I}$. As a performance measure of the empirical basis $\gamma_1, \gamma_2, \ldots, \gamma_M$ we may use $\mathrm{E}||\zeta_M||^2$ and calculate the value of the cross-validation criterion:

$$\mathrm{CV}(\alpha) = \sum_{M=1}^{\infty} \mathrm{CV}_M(\alpha) = \sum_{M=1}^{\infty} \sum_{i=1}^{N} ||\zeta_m^{[i]}(\alpha)||^2 \tag{4.17}$$

where $\zeta_m^{[i]}$ is the part of x_i orthogonal to the subspace spanned by estimates $\hat{\gamma}_1, \hat{\gamma}_2, \ldots, \hat{\gamma}_M$ estimated from the data set with excluded i-th observation. In the next step we will choose the α^{opt} such that:

$$\alpha^{opt} = \arg\min \mathrm{CV}(\alpha)$$

In practice, we will use just the truncated version of the first sum 4.17 – given the data set containing N functions, we are able to estimate only first $N - 1$ eigenfunctions, we may also expect that the eigenfunctions of high "order" are estimated with a large error.

Algorithm

This algorithm is partially taken from Ramsay and Silverman (1997).

1. Center the data x_i (subtract the sample mean).
2. for a given α calculate the smoothed eigenfunctions $\hat{\zeta}_m^{[i]}$ without using the $i-th$ observation. Calculate $\mathrm{CV}(\alpha)$ by truncating at m.
3. Minimize $\mathrm{CV}(\alpha)$ with respect to α.

Using our notation and functional basis expansion we can calculate all the elements of $\mathrm{CV}(\alpha)$:

$$\begin{aligned} \widehat{(\mathbf{G})}_{ij} &= \langle \hat{\gamma}_i, \hat{\gamma}_j \rangle = \mathbf{b}_i^\top \mathbf{W} \mathbf{b}^j \\ \widehat{\langle \gamma_j, x_i \rangle} &= \mathbf{b}_j \mathbf{W} \mathbf{c}_i. \end{aligned}$$

Thus, we can calculate the coefficients of the projection of x onto the subspace spanned by the (estimated) eigenfunctions, denoted by $\mathbf{d}$, clearly

$$\| \zeta_M \|^2 = \left((\mathbf{c} - \mathbf{d})^2 \right)^\top \operatorname{diag}(\mathbf{W}) .$$

The advantage of this algorithm is that is fully data driven. However, it is known from practice that the Cross Validation leads to unstable results. Another disadvantage of this method are high computational costs.

5 Common Principle Components

In the following chapter we will discuss the possibility of implementation of Common Principal Components (CPC) model into the functional framework. In the Section 5.1 we introduce the CPC in the Multivariate model.

5.1 Multivariate common principal components model

The Common Principal Components model (CPC) in the multivariate setting can be motivated as the model for similarity of the covariance matrices in the k-sample problem. Having k random vectors, $\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \ldots, \mathbf{x}_{(k)} \in \mathbb{R}^T$ the CPC-Model can be written as:

$$\boldsymbol{\Psi}_j \stackrel{\text{def}}{=} \text{Cov}(\mathbf{x}_{(j)}) = \boldsymbol{\Gamma}\boldsymbol{\Lambda}_\mathbf{j}\boldsymbol{\Gamma}^\top,$$

where $\boldsymbol{\Gamma}$ is orthogonal matrix and $\boldsymbol{\Lambda}_j = \text{diag}(\lambda_{i1}, \ldots, \lambda_{iT})$. That means that eigenvectors are the same across samples, and just the eigenvalues (variances of the principal components scores) differ.

Using the normality assumption, the sample covariance matrices $\mathbf{S}_j, j = 1, \ldots, k$, are Wishart-distributed:

$$\mathbf{S}_j \sim W_T(N_j, \boldsymbol{\Psi}_j/N_j),$$

and the CPC model can be estimated using ML-estimation with likelihood-function:

$$\text{L}(\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2, \ldots, \boldsymbol{\Psi}_k) = C \prod_{j=1}^{k} \exp\left\{ \text{tr}\left(-\frac{N_j}{2} \boldsymbol{\Psi}_j^{-1} \mathbf{S}_j \right) \right\} (\det \boldsymbol{\Psi}_\mathbf{j})^{-N_j/2},$$

as shown in Flury (1988). Here C is a factor that does not depend on the parameters and N_j is the number of observations in group j. The maximization of this likelihood-function is equivalent to:

$$\prod_{j=1}^{k} \left\{ \frac{\det \text{ diag}(\boldsymbol{\Gamma}^\top \mathbf{S}_\mathbf{j} \boldsymbol{\Gamma})}{\det(\boldsymbol{\Gamma}^\top \mathbf{S}_\mathbf{j} \boldsymbol{\Gamma})} \right\}^{N_j} \tag{5.1}$$

and the maximization of this criterion is performed by the so called FG-algorithm (Flury-Gautschi). This procedure is often referred to as the simultaneous diagonalization. The FG-Algorithm is implemented in the XploRe through the quantlet `CPCFGalg`, the complete estimation for the Common PCA model can be performed using the quantlet `CPC`:

```
{B,betaerror,lambda,lambdaerror,psi} =CPC(A,N)
```

CPC computes the common eigenmatrix, eigenvalues, corresponding standard errors and estimated population covariance matrices from sample covariances of k groups using maximum likelihood.

The input parameters:

`A` $T \times T \times k$ array of k covariances

`N` $k \times 1$ vector of weights, usually number of observations in group k

Output variables:

`B` $T \times T$ matrix, common orthogonal transformation matrix

`betaerror` $T \times T$ matrix, standard errors of B

`lambda` $T \times k$ matrix, T eigenvalues of each group k

`lambdaerror` $T \times k$, T standard errors of eigenvalues of each group k

`psi` $T \times T \times k$ array of k estimated population covariance matrices

5.1.1 FG algorithm

The FG algorithm is a modification of the well known Jacobi (rotation) algorithm. This algorithm is essential for this and the next section, thus let us describe it briefly. As mentioned, the FG algorithm focuses on the minimalization of (5.1). Let us consider a more general case that will be used also in the Section 5.2 and focus on the k

positive definite symmetric matrices $\mathbf{F}_1$,...,$\mathbf{F}_k$. Define the individual "measure of non-diagonality":

$$\Psi(\mathbf{F}_i) = \frac{\det(\text{diag}\mathbf{F}_i)}{\det\mathbf{F}_i}. \tag{5.2}$$

The heuristics of this measure is clear: compare the diagonal and diagonal + nondiagonal terms of a matrix. It is also easy to see that the equality $\Psi(\mathbf{F}_i) = 1$ occurs when the $\mathbf{F}_i$ is already diagonal. From the Theorem B.1 Flury (1988) (Hadamard's equation) follows that for $\mathbf{F}_i$ positive definite is $\Psi(\mathbf{F}_i) \geq 1$. As an overall measure for all k matrices we use an analogue of (5.1):

$$\Psi(\mathbf{F}_1, \ldots, \mathbf{F}_k, N_1, \ldots, N_k) = \prod_{j=1}^{k} \left\{ \frac{\det \text{ diag}(\mathbf{F}_i)}{\det(\mathbf{F}_i)} \right\}^{N_j}, \tag{5.3}$$

a weighted product of the individual measures of non-diagonality. For a given orthogonal matrix $\mathbf{G}$ define $\mathbf{A}_i \stackrel{\text{def}}{=} \mathbf{G}\mathbf{F}_i\mathbf{G}^\top$. Then

$$\Psi_{min}(\mathbf{A}_i, N_i, i = 1, \ldots, k) = \min \left\{ \Psi(\mathbf{G}^\top \mathbf{A}_i \mathbf{G},\ N_i,\ i = 1, \ldots, k) \right\} \tag{5.4}$$

where the minimum is taken with respect to the group of orthogonal matrices of suitable dimension. Then Ψ_{min} can be defined as the best reachable simultaneous diagonalization and Flury (1988) defines this the "measure of simultaneous diagonalizability". The CPC-hypothesis will yield $\Psi_{min} = 1$ – all matrices can be diagonalized with same matrix

$$\mathbf{G}_{min} \stackrel{\text{def}}{=} \arg\min \left\{ \Psi(\mathbf{G}^\top \mathbf{A}_i \mathbf{G},\ N_i\ i = 1, \ldots, k) \right\},$$

once again, the minimum is taken over the group of orthogonal matrices of suitable dimension. Using the first order condition and after some straightforward calculations it can be shown (for details see Flury (1988), Chapter 4) that for $(\mathbf{g}_1, \ldots, \mathbf{g}_k) = \mathbf{G}_{min}$ holds:

$$\mathbf{g}_m^\top \left(\sum_{i=1}^{k} N_i \frac{\lambda_{im} - \lambda_{ij}}{\lambda_{im}\lambda_{ij}} \mathbf{A}_i \right) \mathbf{g}_j = 0, \text{ for } 1 \leq m < j < T \tag{5.5}$$

where T is the dimension of $\mathbf{F}_i$ and

$$\lambda_{il} \stackrel{\text{def}}{=} \mathbf{g}_l^\top \mathbf{A}_i \mathbf{g}_l, \text{ for } i = 1, \ldots, k,\ l = 1, \ldots, T \tag{5.6}$$

The FG algorithm solves these equations iteratively. It consists of two separate algorithms, Algortithm F and G. For initialization of this iterations we need a first approximation $\mathbf{G}^0$ for $\mathbf{G}$, in general we may set $\mathbf{G}^0 = \mathbf{I}_T$.

F1 Do for all (m, j) $1 \leq m < j < T$:
Set $\mathbf{T}_i = (\mathbf{g}_m, \mathbf{g}_j)^\top \mathbf{A}_i (\mathbf{g}_m, \mathbf{g}_j),\ i = 1, \ldots k,$

F1$_1$ Perform the G algorithm on $\mathbf{T_1}, \ldots, \mathbf{T_k}, N_1, \ldots, N_k$:

G0 Initialize G algorithm by a proper matrix $\mathbf{Q}^0$, such as $\mathbf{Q}^0 \stackrel{\text{def}}{=} \mathbf{I}_T$.

G1 Compute $\delta_{ij} \stackrel{\text{def}}{=} \mathbf{q}_j^{g\top} \mathbf{T}_i \mathbf{q}_j^g, i = 1, \ldots, k,\ j = 1, 2$

Set $\mathbf{T} \stackrel{\text{def}}{=} \sum_{i=1}^{k} N_i \frac{\delta_{i1} - \delta_{i2}}{\delta_{i1}\delta i2} \mathbf{T}_i$

G2 Compute the rotation matrix $\mathbf{Q}^{g+1} \stackrel{\text{def}}{=} \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix}$

G3 if $\parallel \mathbf{Q}^{g+1} - \mathbf{Q}^g \parallel_T < \varepsilon_G$ stop, else $g \leftarrow g + 1$ and go to G1. The $\parallel \bullet \parallel_T$ denotes a matrix norm and ε_G a (small) constant.

F1$_3$ Set $\mathbf{J} \stackrel{\text{def}}{=} \mathbf{Q}^{g+1}$.

F1$_3$ Put $\mathbf{G}^f \leftarrow \mathbf{G}^f \mathbf{J}$

F2 if $|\Psi(\mathbf{G}^f) - \Psi(\mathbf{G}^{f-1})| < \varepsilon_f$, where $\varepsilon_f > 0$ then stop, else $f \leftarrow f + 1$ and go to F1

Steps with notation F belongs to F-part of the algorithm and the G noted parts to the G algorithm.

It can be seen that the F part of the algorithm consist of sequential rotation of all $T(T-1)/2$ pairs of column vectors of $\mathbf{G}$, the G part of the algorithm solves by iterations the analog of (5.5):

$$\mathbf{q}_1^\top \left(\sum_{i=1}^{k} N_i \frac{\delta_{i1} - \delta_{i2}}{\delta_{i1}\delta_{i2}} \mathbf{T}_i \right) \mathbf{q}_2 = 0$$

for the 2×2 matrix that is currently changed by F-part.

The proof of convergence of the FG algorithm can be found in Flury (1988), but the existence of the Ψ_{min} is given by the fact that the group of the orthogonal matrices of dimension T is compact.

The analogy of F- and Jacobi-algorithm should be obvious.

The FG algorithm is implemented in the XploRe through the quantlet `CPCFGalg`:

```
{B,Phi} =CPCFGalg(A,N)
```

CPCFGalg implements the FG-Algorithm which finds a common orthogonal transformation matrix in order to simultaneously diagonalize several positive definite symmetric matrices.

The input parameters:

`A` $T \times T \times k$ array of k positive definite symmetric matrices

`N` $k \times 1$ vector of weights, usually number of observations in group k

Output variables:

`B` $T \times T$ matrix, common orthogonal transformation matrix

`Phi` scalar, measure of deviation from diagonality. Phi is 1, corresponds to the complete diagonalibility of the matrices in A. Phi increases monotically in "deviation from diagonality".

5.2 Functional common principal components model

As shown in Section 4, using the advantage of the functional basis expansion, the FPCA and SPCA are basically implemented via the spectral decomposition of the "weighted" covariance matrix of the coefficients. In view of the minimization property of the FG algorithm, the diagonalization procedure optimizing the criterion (5.1) can be employed. However, the estimates obtained may not be maximum likelihood estimates.

We will introduce the ideas and notation on the two sample problem, the generalization to the k-sample problem is straightforward.

Assume we have two functional populations from stochastic processes denoted X^1, X^2, holding the same notation as in the previous chapter $\mathcal{X}_f^1$, $\mathcal{X}_f^2$. Denote the eigenfunctions of the covariance operator of X^1, X^2 by $\gamma_1^1, \gamma_2^1, \ldots$ and $\gamma_1^2, \gamma_2^2 \ldots$ respectively. Using the same functional basis $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_L)^\top$ we may estimate coefficient matrices $\mathbf{C}^1$, $\mathbf{C}^2$. Using the estimation technique of eigenfunction introduced in the Section 4.2 we need to perform the spectral analysis of matrices :

$$\begin{aligned} \mathbf{SWCWS}^1 &\stackrel{\text{def}}{=} \{\mathbf{SW}\,\mathrm{Cov}(\mathbf{C}^1)\mathbf{WS}^\top\}, \\ \mathbf{SWCWS}^2 &\stackrel{\text{def}}{=} \{\mathbf{SW}\,\mathrm{Cov}(\mathbf{C}^2)\mathbf{WS}^\top\}. \end{aligned}$$

Assuming the same eigenfunctions

$$\gamma_1^1 = \gamma_1^2, \gamma_2^1 = \gamma_2^2, \ldots, \tag{5.7}$$

we may use the simultaneous diagonalization technique (FG) algorithm from the multivariate common principal components analysis in order to obtain "common" eigenfunctions.

5.2.1 XploRe implementation

This procedure is implemented in XploRe via quantlet `fdacpcaK`

```
{fpcaresult,values,varprop} =fdacpcaK(CovC, N, basisfd
                            {,lambda, lfd,npc,norm})
```

performs the common smoothed functional PCA, using simultaneous diagnalization

The input parameters:

`CovC` L × L × k array of covariances of coefficients

`N` k × 1 vector of weight, usually number of observations in each group

`basisfd` list, object of type basisfd

`lambda` scalar, the smoothing parameter, default is 0,

`npc` scalar, the number of (generalized) eigenfunctions, default is 4.

`norm` string, normalization type, if norm="orthonorm" fpcaresult.coef are orthonormalized (with respect to the basis penalty matrix), if norm="norm" the coefficients are renormalized to norm=1, default is no normalization

The output variable

`fcpcaresult` list, functional object

`values` × k matrix npc, the (generalized) eigenvalues,

`varprop` npc × k matrix, the proportion of variance explained by the eigenfunctions,

It can be seen that the input and output parameters of the `fdacpcaK` quantlet and `fdaspca` differ – instead of giving the functional object as an input parameter as it is the case in `fdaspca` we use directly the array `CovC`, with covariance matrices of coefficient for k groups and `basisfd` for identification of the functional basis. This solution yields a quantlet that is universal for any number of groups k. To explain

how to use this quantlet and its distinction from the quantlet `fdaspca` we comment this small simulated example:

```
library("fda")
N=30
T=35
Nbasis=20
x=matrix(N)'.*grid(0,1/T,T)
beta2=normal(N)'.*matrix(T)
lambda2=sqrt(0.3)
beta1=normal(N)'.*matrix(T)
lambda1=sqrt(0.6)
y=lambda1*beta1.*sin(x*2*pi)/sqrt(1/2)
  +lambda2*beta2.*cos(x*2*pi)/sqrt(1/2)+0.5*normal(T,N)
fdb = createfdbasis ("fourier", #(0,1),Nbasis,1)
tvec=grid(0,1/T,T)
fd=data2fd(y,tvec,fdb)

x=matrix(N)'.*grid(0,1/T,T)
beta2=normal(N)'.*matrix(T)
lambda2=sqrt(0.6)
beta1=normal(N)'.*matrix(T)
lambda1=sqrt(0.9)
y2=lambda1*beta1.*sin(x*2*pi)/sqrt(1/2)
   +lambda2*beta2.*cos(x*2*pi)/sqrt(1/2)+0.5*normal(T,N)
fdb = createfdbasis ("fourier", #(0,1),Nbasis,1)
fd2=data2fd(y2,tvec,fdb)
```

In this part we create two functional objects `fd` and `fd2`. In `fd` we have 30 observations of simulated functions

$$y_{1i}(t) = \lambda_1^1 \beta_{1i}^1 \frac{\sin(t2\pi)}{\sqrt{1/2}} + \lambda_2^1 \beta_{2i} \frac{\cos(t2\pi)}{\sqrt{1/2}} + 0.5\varepsilon_i^1$$

where $\lambda_1^1 = \sqrt{0.6}$, $\lambda_2^1 = \sqrt{0.3}$, $\beta_1^1, \beta_2^1, \varepsilon_i^1 \sim N(0,1)$.
And `fd2` with 30 observation of

$$y_{2i}(t) = \lambda_1^2 \beta_{1i}^2 \frac{\sin(t2\pi)}{\sqrt{1/2}} + \lambda_2^2 \beta_{2i} \frac{\cos(t2\pi)}{\sqrt{1/2}} + 0.5\varepsilon_i^2$$

where $\lambda_1^2 = \sqrt{0.9}$, $\lambda_2^2 = \sqrt{0.6}$, $\beta_1^2, \beta_2^2, \varepsilon_1^2 \sim N(0,1)$. Thus two data sets with the same eigenfunctions $(\sin, \cos)$ but different eigenvalues. The functions are approximated by

20 Fourier functions from the equidistant grid with 35 points from $[0, 1]$, this choice yields clearly undersmoothed functions. The correct choice here would be clearly 3 Fourier functions, due to the fact that the simulated functions are simply linear combination of sine and cosine functions. The ε's simulate the observation error, it should be denoted we have neglected this error in the the theoretical part of FPCA. We are "contaminating" our data set with this additional error and using $L = 20$ in order to obtain a visible difference in the eigenfunctions for each group. Clearly, this example is slightly artificial, our main aim is to explain the usage of the quantlet `fdacpcaK`.

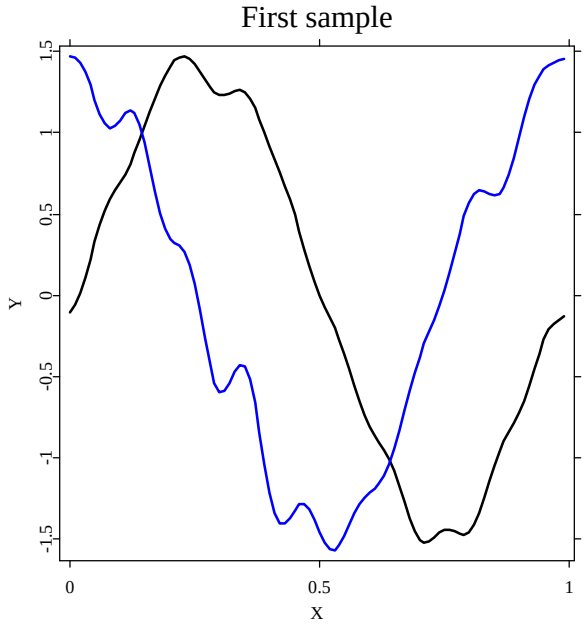

Figure 5.1: Weight functions for the first simulated sample.

`FDAmscCPC.xpl`

```
fpc=fdaspca(fd,0,0,2)
fpc2=fdaspca(fd2,0,0,2)
 Tempdi1=createdisplay(1,1)
 Tempdi2=createdisplay(1,1)
 grpf=grfd(fpc.fpcaresult,grid(0,1/100,100),0,#(0,1))
 show(Tempdi1,1,1,grpf)
 grpf=grfd(fpc2.fpcaresult,grid(0,1/100,100),0,#(0,1))
 show(Tempdi2,1,1,grpf)
```

In this part we use the quantlet `fdaspca` for each sample and obtain first two individual eigenfunctions. The inputs are the functional objects directly. Afterwards we plot the eigenfunctions in separated displays. The plots are displayed in Figures 5.1, 5.2.

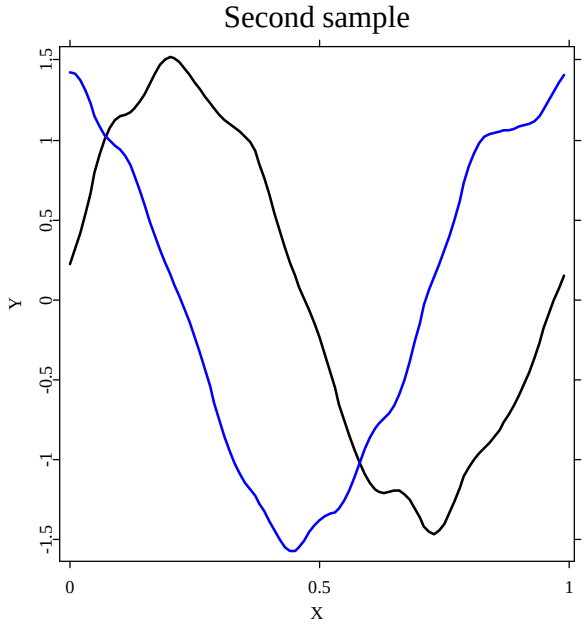

Figure 5.2: Weight functions for the second simulated sample.

FDAmscCPC.xpl

```
covfd=cov(fd.coef')
covfd2=cov(fd2.coef')
cov=stack(covfd,covfd2)
N=30|30
cpc=fdacpcaK(cov,N,fdb,0,0,2)
```

In the part of the quantlet above we create first the covariance matrices of the coefficients of both groups `covfd` and `covfd2` and create a 3-Dimensional array `cov` containing `covfd`, `covfd2`. This is the input for the `fdacpcaK` quantlet.

```
Tempdic=createdisplay(1,1)
grpfc=grfd(cpc.fcpcaKresult,grid(0,1/100,100),0,#(0,1))
show(Tempdic,1,1,grpfc)
```

In the last part of our example we plotted the common eigenfunctions, displayed in the Figure 5.3. Please note that the common estimator is smoother than the estimator of both groups. This is caused by the fact that the common estimation uses the observation of both groups.

5.2.2 Summary and outlook

There are some issues and possible generalizations that need to be discussed in connection with this approach. Firstly using the combination between the functional basis approach and FG-algorithm we assume that the matrices that are diagonalized have same interpretation. It seems to be reasonable to use the same number of functions in the pre-specified basis for the both groups, similarly if we use SPCA we should use same smoothing parameter α for both groups. There is another important difference between the multivariate and functional PCA. In the multivariate case $\mathbb{R}^T$, the number of eigenvalues r is less than T, in the functional framework this would correspond to the $r = \infty$. On the other hand the maximum number of estimated eigenvalues of the sample covariance operator will be N. Using the functional basis approach we get $r \leq L$ as discussed in the previous sections.

The full CPC hypothesis 5.7 may be seen as too restrictive. The reason for this has already been discussed – the number of eigenfunctions obtained using the functional basis technique in the functional data analysis is less or equal to the dimension of the used functional basis. The higher order functions are in real applications driven more by the sampling error then explaining the structure of x_i. From the practical point of view we may assume same dimension in the both functional samples – more formally assuming the existence of $Q \in \mathbb{N}$, $\lambda_j^1 = \lambda_j^2 = 0,\ j > Q$. Or alternatively we may

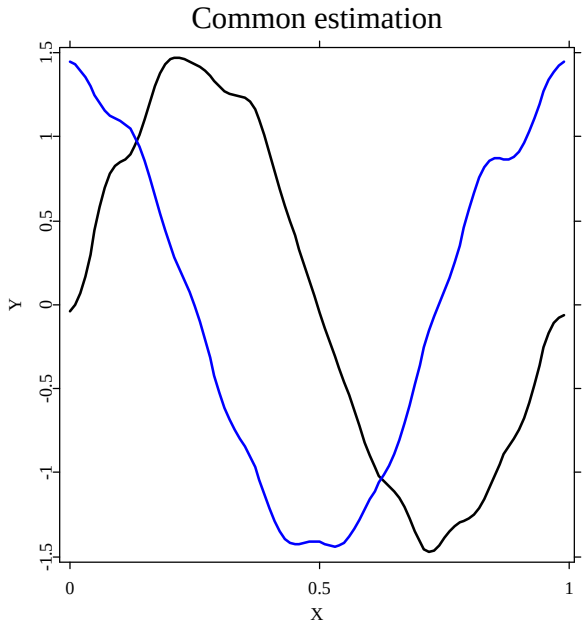

Figure 5.3: Weight functions (common) estimated from both simulated samples.
FDAmscCPC.xpl

assume just the equality of first R eigenfunctions:

$$\gamma_j^1 = \gamma_j^2 = \ldots, \gamma_j^K, \ j = 1, \ldots, R, \tag{5.8}$$

that corresponds to the Partial Common Principal Hypothesis as referred to in Flury (1988). In this framework we have first R common eigenfunctions and remaining eigenfunctions are specific for each group. For this model an modified estimation procedure for the partial simultaneous diagonalization. This procedure is implemented in XploRe via the quantlet `CPCp`. The quantlet `fdaspcaK` can easily be changed for this model. Another generalization may be motivated by the fact that, the principal components are often used as the factors in the regression models, see Fengler, Härdle and Villa (2003) among others. The common principal components hypothesis yield the same factor functions for different samples that reduces the dimensionality of the problem rapidly. However, for this problem it might be more interesting not to test just the equality of the eigenfunctions (5.7) but rather the equality of the "eigenspaces"

– spaces spanned by the first R eigenfunctions: $L_R^k \stackrel{\text{def}}{=} \text{span}\{\gamma_j^1,\ j = 1, \dots R\}\ k = 1, \dots, K,$ for some R.

In this chapter we have shown a procedure for the estimation under the common eigenfunction hypothesis. However, the hypothesis of common eigenfunctions needs to be statistically tested. This is still a open question for further research. An possible approaches here can be given by the asymptotical results for eigenvalues and eigenfunctions given by Dauxois, Pousse and Romain (1982).

Another interesting question in the two-sample (analogously in k-sample) problem is the hypothesis for equality of each eingenvalue $\lambda_j = 1 = \lambda_j^2$. The test for this hypothesis can be based on the fact that the $\lambda_j^1 = \text{Var}[\beta_j^1]$ and $\lambda_j^2 = \text{Var}[\beta_j^2]$, where β_j^1 and β_j^1 are the principal components scores of first and second group respectively. Takeda and Sugiyama (2003) proposed a permutation test for the equality of this variance as the testing procedure for the hypothesis of "common" eigenvalues.

6 XploRe quantlet lists

In this chapter we present list of the quantlets which have been created for the new XploRe library `fda` by author. The complete library consist of the quantlets presented here and the quantlets created by Mr. Ulbricht, these are not presented in this thesis. The original quantlets for FDA in XploRe written by `FDApca` and `Fouriertrans` and `Fouriereval` have been completely rewritten, however due to the backward compatibility we keep this quantlets in the XploRe system. For comparison of the implementation we refer to the packages for R,S, Matlab created by Ramsay (2003).

We present the quantlets in the short form, without the help templates.

6.1 Basic statistics

`fdamean`

```
proc(fdamean)=fdamean(fdobject)
  error(names(fdobject)!=("coef"|"basisfd"),"FDAmean: Input is not
    valid fdobject")
  coef=mean(fdobject.coef')'
  basisfd=fdobject.basisfd
  fdamean=list(coef,basisfd)
endp
```

`evalfdavar`

```
proc(fdvar)=evalfdavar(evalarg,fdobject)

 error(names(fdobject)!=("coef"|"basisfd"),"evalfdavar: Input is not
   valid fdobject")
 error(((min(evalarg) < fdobject.basisfd.range[1]) || (max(evalarg)>
   fdobject.basisfd.range[2])),"evalfdavar: elemets of EVALARG out of
   basisfd.range")
 n=rows(evalarg)
 evalmat=getbasismatrix(evalarg,fdobject.basisfd)
 C=cov(fdobject.coef')
 i=1
```

```
 fdvar=NaN
 while(i<=n)
 fdvar=fdvar|(evalmat[i,]*C*trans(evalmat[i,]))
 i=i+1
 endo
 fdvar=fdvar[2:n+1]
endp
```

evalfdacov

```
proc(fdcov)=evalfdacov(evalarg,fdobject)

 error(names(fdobject)!=("coef"|"basisfd"),"evalfdacov: Input is not
   valid fdobject")
 error(((min(evalarg) < fdobject.basisfd.range[1]) || (max(evalarg)>
   fdobject.basisfd.range[2])),"evalfdacov: elemets of EVALARG out of
   basisfd.range")
 n=rows(evalarg)
 evalmat=getbasismatrix(evalarg,fdobject.basisfd)
 C=cov(fdobject.coef')
 nresol=rows(evalarg)
 ti=grid(#(1,1),#(1,1),#(nresol,nresol))
 i=1
 n=rows(ti)
 ti=ti~NaN*matrix(n)
 while(i<=n)
 ti[i,]=evalarg[ti[i,1]]~evalarg[ti[i,2]]~(evalmat[ti[i,1],]*C*trans(
   evalmat[ti[i,2],]))
 i=i+1
 endo
 fdcov=ti
 ti=0
endp
```

evalfdacorr

```
proc(fdcorr)=evalfdacorr(evalarg,fdobject)

 error(names(fdobject)!=("coef"|"basisfd"),"evalfdacorr: Input is not
   valid fdobject")
 error(((min(evalarg) < fdobject.basisfd.range[1]) ||
   (max(evalarg)>fdobject.basisfd.range[2])),"evalfdacorr: elemets of
     EVALARG out of basisfd.range")
 n=rows(evalarg)
 evalmat=getbasismatrix(evalarg,fdobject.basisfd)
 C=cov(fdobject.coef')
 nresol=rows(evalarg)
 ti=grid(#(1,1),#(1,1),#(nresol,nresol))
 i=1
 n=rows(ti)
 ti=ti~NaN*matrix(n)
 while(i<=n)
```

```
 ti[i,]=evalarg[ti[i,1]]~evalarg[ti[i,2]]~(evalmat[ti[i,1],]*C*trans(
   evalmat[ti[i,2],]))
       /sqrt((evalmat[ti[i,1],]*C*trans(evalmat[ti[i,1],]))*(evalmat[
         ti[i,2],]*C*trans(evalmat[ti[i,2],])))
 i=i+1
 endo
 fdcorr=ti
 ti=0
endp
```

6.1.1 Functional principal components

fdaspca

```
proc(fpcaresult,values,varprop,scores)=fdaspca(fdobject,lambda,lfd,npc,
  orthonorm)

  error((exist("fdobject").<>9),"fdaspca: No fdobject,or fdaspca not a
    list!")
  if (exist("lambda").<>1)
    lambda = 0             ; Default for the lambda
  endif
  error(lambda<0,"fdaspca: Negative lambda is not possible!")
  if (exist("norm").<>2)
    norm="no"
  endif
  if (exist("lfd").<>1)
    lfd=2                                ; Default for lfd
  endif
  if (exist("npc").<>1)
    npc = 4             ; Default for the number of PCs is 4
  endif
  K = rows(fdobject.coef)        ; number of Basis f
  N = cols(fdobject.coef)        ; number of observ
;centring (is not neccesery because cov used)
  fdobject.coef=trans(fdobject.coef'- mean(fdobject.coef'))
  Vmat = cov(fdobject.coef')     ; covariance matrix of coefficients
  Jmat = inprod(fdobject.basisfd,fdobject.basisfd,0,0); identiy matrix
    for penalty matrix
  if (lambda == 0) ; ordinary FPCA
    Jmat=(Jmat + Jmat')/2

;Cholesky factor:
    Lmatch= chold(Jmat,K)
    di=diag(xdiag(Lmatch))
    bi=Lmatch-di+eye(K)
    Lmat=(bi'*sqrt(di))'

    Lmatinv=inv(Lmat)
    Vmat=Lmat*Vmat*Lmat'
```

```
    Vmat=(Vmat + Vmat')/2

; eigenanalysis

    result = eigsm(Vmat)               ; eigenvalue problem
    rest=result.values~trans(result.vectors) ; sort eigval
    rest=sort(rest,-1)
    result.values=rest[,1]
    result.vectors=orthonormal(trans(rest[,2:cols(rest)]))

    vectors = Lmatinv*result.vectors
    values=result.values
    rest=0

  else ; regularized (smoothed) FPCA
    Kmat=inprod(fdobject.basisfd,fdobject.basisfd,lfd,lfd)
    LLmat= Jmat + lambda * Kmat
    LLmat=(LLmat + LLmat')/2

;Lmat = chold(LLmat,K)               ; Cholesky decomposition
;Cholesky factor: correct
    Lmatch= chold(LLmat,K)
    di=diag(xdiag(Lmatch))
    bi=Lmatch-di+eye(K)
    Lmat=(bi'*sqrt(di))'

    Lmatinv = inv(Lmat)
    Vmat= Lmatinv'*Jmat*Vmat*Jmat*Lmatinv
    Vmat = (Vmat+Vmat')/2              ; symmetrizing

; eigenanalysis

    result = eigsm(Vmat)               ; eigenvalue problem
    rest=result.values~trans(result.vectors)
    rest=sort(rest,-1)
    result.values=rest[,1]
    result.vectors=orthonormal(trans(rest[,2:cols(rest)]))
    values = result.values             ; eigenvalues
    vectors = Lmatinv*result.vectors

  endif

  coef = vectors[,1:npc]; coefficient of eigenfunctions

  if(norm=="orthonorm") ; orthonormalization with resp to Jmat
    n=npc
    i=1
    while(i<=n)
      coef[,i]=coef[,i]/sqrt((coef[,i]')*Jmat*(coef[,i]))
      j=i+1
      while(j<=n)
        coef[,j]=coef[,j]-(coef[,j]'*Jmat*coef[,i])*coef[,i]
```

```
        j=j+1
      endo
      i=i+1
    endo
  endif

 if(norm=="norm"); rescaling
    n=npc
    i=1
    while(i<=n)
      coef[,i]=coef[,i]/sqrt((coef[,i]')*Jmat*(coef[,i]))
      i=i+1
    endo
  endif

  varprop  = values/sum(values)      ; proportion of each variance
  values   = values[1:npc]           ; selected eigenvalues
  varprop  = varprop[1:npc]          ; selected proportion
  scores   = fdobject.coef' * Jmat * coef ; PC scores
  fpcaresult=list(coef,fdobject.basisfd)
  endp
```

fdacpcaK

```
  p=dim(CovC)[1]            ; covariances are pxp
  k=dim(CovC)[3]            ; number of groups

;input parameter check CPC part
  error (p!=dim(CovC)[2],"fdacpcaK: Covariance matrices are not
    quadratic!")
  error (sum(sum(sum( abs(CovC.-CovC'  != 0), 3), 2)) != 0,"fdapcaK:
    Covariance matrices are not symmetric!")
  error (k!=rows(N),"fdacpcaK: Number of matrices not compatible with
    number of weights!")
  error (sum(N<=0),"fdacpcaK: Weights cannot be zero or negative!")

;input parameter check FDA part + defaults
  error((exist("basisfd").<>9),"fdacpcaK: No basisfd,or basisfd not a
    list!")
  error(basisfd.nbasis<>p,"fdacpcaK: rows(CovC)<>basisfd.nbasis!"); the
     number of cofs in CovC <> number of cofs in basisfd

  if (exist("lambda").<>1)
    lambda = 0             ; Default for the lambda
  endif
  error(lambda<0,"fdacpcaK: Negative lambda is not possible!")
  if (exist("norm").<>2)
    norm="no"
  endif
  if (exist("lfd").<>1)
    lfd=2                              ; Default for lfd
  endif
```

```
  if (exist("npc").<>1)
    npc = 4             ; Default for the number of PCs is 4
  endif

 K=rows(CovC) ; number of basis functions (=dim of CovC)

  Jmat = inprod(basisfd,basisfd,0,0); identiy matrix for penalty matrix
  if (lambda == 0) ; ordinary FPCA
    Jmat=(Jmat + Jmat')/2

;Cholesky factor:
    Lmatch= chold(Jmat,K)
    di=diag(xdiag(Lmatch))
    bi=Lmatch-di+eye(K)
    Lmat=(bi'*sqrt(di))'
    Lmatinv=inv(Lmat)
;Creating the final array for CPCFGAlg
    j=1
    while(j<=k)
      CovC[,,j]=Lmat*CovC[,,j]*Lmat'
      j=j+1
    endo
    CovC=(CovC + CovC')/2
    B=CPCFGalg(CovC,N){1}

; Calculating the eigenvalues in lambda
    lam=reshape(xdiag((B'.*matrix(p,p,k))*CovC*(B.*matrix(p,p,k))),p|k)


; sort eigenvectors according to size of its corresponding eigenvalues
    u=lam'|B
    u=sort(u',-1)
    lam=u[,1:k]
    B=u[,k+1:k+p]'

    coef =Lmatinv*B

  else ; regularized (smoothed) FPCA
    Kmat=inprod(basisfd,basisfd,lfd,lfd)
    LLmat= Jmat + lambda * Kmat
    LLmat=(LLmat + LLmat')/2

  ;Lmat = chold(LLmat,K)               ; Cholesky decomposition
 ;Cholesky factor: correct
    Lmatch= chold(LLmat,K)
    di=diag(xdiag(Lmatch))
    bi=Lmatch-di+eye(K)
    Lmat=(bi'*sqrt(di))'

    Lmatinv = inv(Lmat)

    ;Creating the final array for CPCFGAlg
```

```
76      j=1
77      while(j<=k)
78       CovC[,,j]= Lmatinv'*Jmat* CovC[,,j]*Jmat*Lmatinv
79        j=j+1
80      endo
81      CovC=(CovC + CovC')/2
82
83    ;CPCFGalg
84      B=CPCFGalg(CovC,N){1}
85
86  ; Calculating the eigenvalues in lambda
87      lam=reshape(xdiag((B'.*matrix(p,p,k))*CovC*(B.*matrix(p,p,k))),p|k)
88
89
90  ; sort eigenvectors according to size of its corresponding eigenvalues
91      u=lam'|B
92      u=sort(u',-1)
93      lam=u[,1:k]
94      B=u[,k+1:k+p]'
95
96     coef=Lmatinv*B ; coefficients of eigenvectors
97
98  endif
99    coef = coef[,1:npc]; coefficient of eigenfunctions
100
101    if(norm=="orthonorm") ; orthonormalization with resp to Jmat
102     n=npc
103     i=1
104     while(i<=n)
105       coef[,i]=coef[,i]/sqrt((coef[,i]')*Jmat*(coef[,i]))
106       j=i+1
107       while(j<=n)
108         coef[,j]=coef[,j]-(coef[,j]'*Jmat*coef[,i])*coef[,i]
109         j=j+1
110       endo
111       i=i+1
112     endo
113   endif
114
115  if(norm=="norm"); rescaling
116     n=npc
117     i=1
118     while(i<=n)
119       coef[,i]=coef[,i]/sqrt((coef[,i]')*Jmat*(coef[,i]))
120       i=i+1
121     endo
122   endif
123
124   fcpcaKresult=list(coef,basisfd)
125   values = lam
126   varprop  = values/sum(values)      ; proportion of each variance
127   values = values[1:npc,]
```

```
  varprop  = varprop[1:npc,]          ; selected proportion
endp
```

6.2 Graphical tools

grfd

```
proc(gfd)=grfd(fd,evalarg,Lfd,col,art,thick)

 if (exist(evalarg) == 0)
  evalarg=grid(fd.basisfd.range[1],(fd.basisfd.range[2]-fd.basisfd.
    range[1])/100,99)
 endif

 if (exist("col").<>1)
    col = 0
  endif

 if (exist("art").<>1)
    art = 1
  endif

 if (exist("thick").<>1)
    thick = 2
  endif

 if (exist (Lfd) == 0)
    Lfd = 0
  endif

  error (cols (evalarg) != 1 && cols (evalarg) != cols (fd.coef), "grfd
    : Argument GRIDT must have columns equal to fd.coef or equal to one
    ")
  error (exist (fd.coef) == 0, "grfd: Argument FD is no fd object")
  error (cols (Lfd) > 2, "grfd: LFD cannot exceed two dimensions!")

  if (rows (dim (fd.coef)) == 3)
    dim3coef = dim (fd.coef)[3]
  else dim3coef = 1
  endif

  if (cols (Lfd) == 2)
    rowsLfd = 1
  else rowsLfd = rows (Lfd)
  endif

  evalmat=evalfd(evalarg,fd,Lfd)

; analogue of grpcp
```

```
  i  = 0
  n  = rows(evalmat)
  p  = cols(evalmat)
  while (i.<p)
    i = i+1
    if (i.=1)
      gfd =evalarg~evalmat[,i]
    else
      gfd = gfd|(evalarg~evalmat[,i])
    endif
  endo
  lt = ((1:n)+trans(grid(0, n, p)))'
  setmaskp (gfd, 0, 0, 0)
  setmaskl (gfd, lt, col, art, thick)
endp
```

The quantlet `grfd` creates the graphical object with all functions if we want to plot just first 10 functions we can simply extract corresponding coefficients, as shown in following example.

```
y = read ("dailtemp.dat")
tvec=#(1:365)/365
fdb = createfdbasis ("fourier", #(0,1),31,1)
fdtempf31=data2fd(y,tvec,fdb)
fdtempf31.coef=fdtempf31.coef[,1:10]
Tempdi=createdisplay(1,1)
grtempf=grfd(fdtempf31)
show(Tempdi,1,1,grtempf)
```

grfdavar

```
proc(gdfvar)=grfdavar(fd,evalarg,col,art,thick)
  if (exist(evalarg) == 0)
    evalarg=grid(fd.basisfd.range[1],(fd.basisfd.range[2]-fd.basisfd.
      range[1])/100,99)
  endif

  if (exist("col").<>1)
    col = 0
  endif

  if (exist("art").<>1)
    art = 1
  endif

  if (exist("thick").<>1)
    thick = 2
  endif
```

```
  error (cols (evalarg) != 1 && cols (evalarg) != cols (fd.coef), "
    grfdavar: Argument GRIDT must have columns equal to fd.coef or
    equal to one")
  error (exist (fd.coef) == 0, "grfdavar: Argument FD is no fd object")
  fdavarf=evalfdavar(evalarg,fd)
  gdfvar=evalarg~fdavarf
  setmaskl(gdfvar,#(1:rows(evalarg))',col,art,thick)
  setmaskp(gdfvar,0,0,0)
endp
```

grfdacov

```
proc(gs)=grfdacov(fdobject,evalarg,col)

 if (exist("col").<>1)
    col = 0
  endif

 error(names(fdobject)!=("coef"|"basisfd"),"grfdacov: Input is not
   valid fdobject")
 error(((min(evalarg) < fdobject.basisfd.range[1]) || (max(evalarg)>
   fdobject.basisfd.range[2])),"grfdacov: elemets of EVALARG out of
   basisfd.range")
 n=rows(evalarg)
 evalmat=getbasismatrix(evalarg,fdobject.basisfd)
 C=cov(fdobject.coef')
 nresol=rows(evalarg)
 ti=grid(#(1,1),#(1,1),#(nresol,nresol))
 i=1
 n=rows(ti)
 ti=ti~NaN*matrix(n)
 while(i<=n)
 ti[i,]=evalarg[ti[i,1]]~evalarg[ti[i,2]]~(evalmat[ti[i,1],]*C*trans(
   evalmat[ti[i,2],]))
 i=i+1
 endo
 fdcov=ti
 ti=0
 gs=grsurface(fdcov,col)
endp
```

grfdacorr

```
proc(gs)=grfdacorr(fdobject,evalarg,col)

if (exist("col").<>1)
    col = 0
  endif
 error(names(fdobject)!=("coef"|"basisfd"),"grfdacorr: Input is not
   valid fdobject")
 error(((min(evalarg) < fdobject.basisfd.range[1]) || (max(evalarg)>
   fdobject.basisfd.range[2])),"grfdacorr: elemets of EVALARG out of
```

```
    basisfd.range")
 n=rows(evalarg)
 evalmat=getbasismatrix(evalarg,fdobject.basisfd)
 C=cov(fdobject.coef')
 nresol=rows(evalarg)
 ti=grid(#(1,1),#(1,1),#(nresol,nresol))
 i=1
 n=rows(ti)
 ti=ti~NaN*matrix(n)
 while(i<=n)
 ti[i,]=evalarg[ti[i,1]]~evalarg[ti[i,2]]~(evalmat[ti[i,1],]*C*trans(
    evalmat[ti[i,2],]))/sqrt((evalmat[ti[i,1],]*C*trans(evalmat[ti[i
    ,1],]))*(evalmat[ti[i,2],]*C*trans(evalmat[ti[i,2],])))
 i=i+1
 endo
 fdcorr=ti
 ti=0

 gs=grsurface(fdcorr,col)
endp
```

6.3 Plotting tools

plotfd

```
proc()=plotfd(fd,evalarg,Lfd,col,art,thick)

 if (exist(evalarg) == 0)
  evalarg=grid(fd.basisfd.range[1],(fd.basisfd.range[2]-fd.basisfd.
    range[1])/100,99)
 endif

 if (exist("col").<>1)
    col = 0
  endif

 if (exist("art").<>1)
    art = 1
  endif

 if (exist("thick").<>1)
    thick = 2
  endif

 if (exist (Lfd) == 0)
    Lfd = 0
  endif

```

```
23   error (cols (evalarg) != 1 && cols (evalarg) != cols (fd.coef), "
       plotfd: Argument GRIDT must have columns equal to fd.coef or equal
       to one")
24   error (exist (fd.coef) == 0, "plotfd: Argument FD is no fd object")
25   error (cols (Lfd) > 2, "plotfd: LFD cannot exceed two dimensions!")
26
27   if (rows (dim (fd.coef)) == 3)
28     dim3coef = dim (fd.coef)[3]
29   else dim3coef = 1
30   endif
31
32   if (cols (Lfd) == 2)
33     rowsLfd = 1
34   else rowsLfd = rows (Lfd)
35   endif
36
37   plotfddi=createdisplay(1,1)
38   grfdt=grfd(fd,evalarg,Lfd,col,art,thick)
39   show(plotfddi,1,1,grfdt)
40
41 endp
```

plotfdavar

```
1  proc()=plotfdavar(fd,evalarg,col,art,thick)
2
3    if (exist(evalarg) == 0)
4      evalarg=grid(fd.basisfd.range[1],(fd.basisfd.range[2]-fd.basisfd.
         range[1])/100,99)
5    endif
6
7    if (exist("col").<>1)
8      col = 0
9    endif
10
11   if (exist("art").<>1)
12     art = 1
13   endif
14
15   if (exist("thick").<>1)
16     thick = 2
17   endif
18   error (cols (evalarg) != 1 && cols (evalarg) != cols (fd.coef), "
       plotfdavar: Argument GRIDT must have columns equal to fd.coef or
       equal to one")
19   error (exist (fd.coef) == 0, "plotfdavar: Argument FD is no fd object
       ")
20
21   fdavarf=evalfdavar(evalarg,fd)
22   gdfvar=evalarg~fdavarf
23   setmaskl(gdfvar,#(1:rows(evalarg))',col,art,thick)
24   setmaskp(gdfvar,0,0,0)
```

```
  plotfddi=createdisplay(1,1)
  show(plotfddi,1,1,gdfvar)
endp
```

plotfdacov

```
proc(gs)=plotfdacov(fdobject,evalarg,col,plot3Ds)

if (exist("col").<>1)
    col = 0
  endif
if (exist("plot3Ds").<>2)
    plot3Ds = "No"
  endif

 error(names(fdobject)!=("coef"|"basisfd"),"plotfdacov: Input is not
   valid fdobject")
 error(((min(evalarg) < fdobject.basisfd.range[1]) || (max(evalarg)>
   fdobject.basisfd.range[2])),"plotfdacov: elemets of EVALARG out of
   basisfd.range")
 n=rows(evalarg)
 evalmat=getbasismatrix(evalarg,fdobject.basisfd)
 C=cov(fdobject.coef')
 nresol=rows(evalarg)
 ti=grid(#(1,1),#(1,1),#(nresol,nresol))
 i=1
 n=rows(ti)
 ti=ti~NaN*matrix(n)
 while(i<=n)
 ti[i,]=evalarg[ti[i,1]]~evalarg[ti[i,2]]~(evalmat[ti[i,1],]*C*trans(
   evalmat[ti[i,2],]))
 i=i+1
 endo
 fdcov=ti
 ti=0
 gs=grsurface(fdcov,col)
 if (plot3Ds=="3D")
 plot3d(3,gs)
 else
 plotdisplay=createdisplay(1,1)
 show(plotdisplay,1,1,gs)
 endif
endp
```

plotfdacorr

```
proc(gs)=plotfdacorr(fdobject,evalarg,col,plot3Ds)

if (exist("col").<>1)
    col = 0
  endif
if (exist("plot3Ds").<>2)
```

```
7     plot3Ds = "No"
8   endif
9
10  error(names(fdobject)!=("coef"|"basisfd"),"plotfdacorr: Input is not
      valid fdobject")
11  error(((min(evalarg) < fdobject.basisfd.range[1]) || (max(evalarg)>
      fdobject.basisfd.range[2])),"plotfdacorr: elemets of EVALARG out of
      basisfd.range")
12  n=rows(evalarg)
13  evalmat=getbasismatrix(evalarg,fdobject.basisfd)
14  C=cov(fdobject.coef')
15  nresol=rows(evalarg)
16  ti=grid(#(1,1),#(1,1),#(nresol,nresol))
17  i=1
18  n=rows(ti)
19  ti=ti~NaN*matrix(n)
20  while(i<=n)
21  ti[i,]=evalarg[ti[i,1]]~evalarg[ti[i,2]]~(evalmat[ti[i,1],]*C*trans(
      evalmat[ti[i,2],]))/sqrt((evalmat[ti[i,1],]*C*trans(evalmat[ti[i
      ,1],]))*(evalmat[ti[i,2],]*C*trans(evalmat[ti[i,2],])))
22  i=i+1
23  endo
24  fdcorr=ti
25  ti=0
26  gs=grsurface(fdcorr,col)
27  if (plot3Ds=="3D")
28  plot3d(3,gs)
29  else
30  plotdisplay=createdisplay(1,1)
31  show(plotdisplay,1,1,gs)
32  endif
33 endp
```

6.4 Summary and outlook

We would like to point out that we are not concerning library `fda` as final, more than as a first step in this project. The usage of the library by different users will surely yield improvements in the user interface. The natural development is to implement different statistical methods for FDA, e.g. functional linear models or functional ANOVA into the existing XploRe-FDA environment.

More challenging are the improvements in the technical part of the library. We are planning to implement refinements in the estimation of the coefficient matrix and support more dimensional functional objects (e.g. surfaces).

Bibliography

Benko, M. and Kneip, A. (2005). Common functional component modeling *Proceedings of ISI 2005*, forthcomming.

Dauxois, J., Pousse, A. and Romain, Y. (1982). Asymptotic Theory for the Principal Component Analysis of a Vector Random Function: Some Applications to Statistical Inference, *Journal of Multivariate Analysis 12*: 136-154.

Flury, B. (1988). *Common Principal Components and Related Models*, Wiley, New York.

Fengler, M., Härdle, W. and Schmidt, P. (2002). Common Factors Governing VDAX Movements and the Maximum Loss, *Journal of Financial Markets and Portfolio Management 16, Nm. 1*: 16-29.

Fengler, M., Härdle, W. and Villa, P. (2003). The Dynamics of Implied Volatilities: A common principle components approach, *Review of Derivative Research 6*: 179-202.

Fengler, M., Härdle, W. and Mammen, E. (2003). Implied Volatitlity String Dynamics, CASE Discussion Paper, http://www.case.hu-berlin.de.

Hastie, T., Tibshirani, R. and Friedman, J., (2002). *The Elements of Statistical Learning*, Springer.

Härdle, W. (1990). *Applied Nonparametric Regression*, Cambridge University Press.

Härdle, W., Müller, M., and Klinke, S. (2001). *XploRe Learning Guide*, Springer, Berlin.

Hafner, R. and Wallmeier, M. (2001). The Dynamics of DAX Implied Volatilities, *International Quarterly Journal of Finance 1, Nm. 1*: 1-27.

Härdle, W. and Simar, L. (2003). *Applied Multivariate Statistical Analysis*, Springer-Verlag Berlin Heidelberg.

Ulbricht, J. (2004). Representing smooth functions, *Master Arbeit, Statistics*, Humboldt Universität zu Berlin.

Kneip, A. and Utikal, K. (2001). Inference for Density Families Using Functional Principal Components Analysis, *Journal of the American Statistical Association 96*: 519-531.

Ramsay, J. (2003). *Matlab, R and S-Plus Functions for Functional Data Analysis*, McGill University, ftp://ego.psych.mcgill.ca/pub/FDAfuns

Ramsay, J. and Silverman, B. (1997). *Functional Data Analysis*, Springer, New York.

Rice, J. and Silverman, B. (1991). Estimating the Mean and Covariance Structure Nonparametrically when the Data are Curves, *Journal of Royal Statistical Society, Ser. B 53*: 233-243.

Silverman, B. (1996). Smoothed Functional Principal Components Analysis by Choice of Norm, *Annals of Statistics 24*: 1-24.

XploRe Inner Group. (2004). XploRe, Auto Pilot Support System, *MD*Tech*, Berlin.

Takeda, Y. and Sugiyama, T. (2003). *Some tests in Functional Principal Components Analysis*, unpublished manuscript, Chuo University.